uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2019-2020

*Trabajo Fin de Grado*

# Desarrollo de un Coche a Escala reducida para Aprendizaje de Conducción Autónoma mediante Técnicas de *Deep Learning*

---


Javier Corrochano Jiménez

Tutor

Juan Manuel Alonso Weber


Leganés, Julio 2020


Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

# ABSTRACT

Technological advances in autonomous driving are growing at a vertiginous rate and are gradually being integrated into society. The possibility of reducing the number of accidents, reducing the effort required to drive and facilitating travel for the disabled are some of the advantages of autonomous vehicles.

In this project, a scale vehicle will be developed that learns to drive autonomously. This will require the combination of a series of important elements, such as the scale vehicle, a driving circuit and a computer with a control device (steering wheel). This setting should allow both supervised driving and autonomous driving learning.

To perform autonomous driving training, an approach divided into two tasks is proposed: extracting features from road lines and learning to drive from these features. For the first task, the construction of an auxiliary scenario (based on chromas) is proposed to facilitate the extraction of the lines. These lines can be virtually superimposed on images of the driving scenario, forming augmented images that will serve to train an Autoencoder. In this way, this network will learn to isolate the lines from the road in real scenarios, making it possible to learn to drive in them.

As a result, a small-scale vehicle is obtained that is capable of driving autonomously along the route of a circuit delimited by two lines of road. It has been observed how the addition of noise during the learning process can improve the robustness of a neural network. The auxiliary scenario has made possible to differentiate the road lines from the rest of the elements of the scene. In this way it has been possible to obtain a vector of features related only to these lines, to generate augmented images of a real scenario and to learn to drive on it.



**Keywords**: Deep Learning, Autonomous driving, Denoising Stacked Convolutional Autoencoder, Behavioral Cloning, Feature Extraction, Scale-Vehicle.

# RESUMEN

Los avances tecnológicos en conducción autónoma crecen a un ritmo vertiginoso y poco a poco se están integrando en la sociedad. La posibilidad de reducir la cantidad de accidentes, disminuir el esfuerzo que hay que dedicar a la conducción y facilitar los desplazamientos a personas discapacitadas son algunas de las ventajas de los vehículos autónomos.

En este proyecto se desarrollará un vehículo a escala que aprenda a conducir de forma autónoma. Para ello será necesario combinar una serie de elementos importantes, como un circuito, un vehículo a escala y un ordenador que procese la información recibida por éste y que permita controlarlo de forma supervisada.

Para realizar el aprendizaje de la conducción autónoma se propone un enfoque dividido en dos tareas: la extracción de características de las líneas de la carretera y el aprendizaje de la conducción a partir de dichas características. Para la primera tarea se propone la construcción de un escenario auxiliar (basado en chromas) que facilite la extracción de las líneas. Dichas líneas se podrán superponer de forma virtual en imágenes del escenario de conducción, formando imágenes aumentadas que servirán para entrenar un *Autoencoder*. De esta forma se conseguirá que esta red aprenda a aislar las líneas de la carretera en escenarios reales, siendo posible realizar el aprendizaje de conducción en ellos.

Como resultado se obtiene un vehículo a escala reducida capaz de conducir de forma autónoma siguiendo el recorrido de un circuito delimitado por dos líneas de carretera. Se ha podido observar como la introducción de ruido durante el proceso de aprendizaje puede mejorar la robustez de una red de neuronas. El escenario auxiliar ha posibilitado diferenciar las líneas de la carretera del resto de elementos de la escena. De esta forma ha sido posible obtener un vector de características relacionado únicamente con dichas líneas, generar imágenes aumentadas sobre un escenario real y realizar el aprendizaje de la conducción sobre él.

**Palabras clave**: *Deep Learning*, Conducción Autónoma, *Denoising Stacked Convolutional Autoencoder*, Imitación del Comportamiento, Extracción de Características, Vehículo a Escala.

## AGRADECIMIENTOS

Deseo expresar mi gratitud hacia aquellas personas que me han acompañado durante toda esta etapa de crecimiento personal y profesional.

En primer lugar quiero agradecer a mi tutor. Gracias por haber confiado en mí para desarrollar este proyecto y por todo el conocimiento, material, orientación, apoyo y tiempo que me has ofrecido.

Gracias a mi familia por haber aguantado mis cambios de humor en estos últimos meses, por su apoyo y por transmitirme el interés por el automovilismo que en parte me ha llevado a realizar este proyecto.

Especialmente, gracias a uno de los pilares fundamentales de mi vida, mi pareja, que me ha sufrido durante estos 4 años y que en todo momento ha tenido paciencia, comprensión y me ha apoyado emocionalmente.

Por último, quiero agradecer a mis amigos por su interés en mi trabajo y sobre todo por su compañía que me ha mantenido en pie.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1. INTRODUCCIÓN

## 1.1. Motivación

Los avances tecnológicos en conducción autónoma crecen a un ritmo vertiginoso y poco a poco se están integrando en la sociedad. La posibilidad de reducir la cantidad de accidentes, disminuir el esfuerzo que hay que dedicar a la conducción y facilitar los desplazamientos a personas discapacitadas son algunas de las ventajas de los vehículos autónomos.

En los últimos años empresas como Google [1], General Motors [2] o Tesla [3] entre otras muchas se han involucrado en la investigación y puesta en marcha de sistemas de conducción autónoma. Los vehículos autónomos comerciales más avanzados controlan la dirección y la velocidad en zonas no urbanas. Aunque sigue siendo necesario que el conductor preste atención en la conducción, los avances tecnológicos y reguladores permitirán la conducción autónoma total en el futuro. De hecho, ya se están realizando pruebas con vehículos casi totalmente autónomos en algunas áreas públicas en EEUU [4].

Hoy en día existen diferentes métodos para abordar estos sistemas de conducción autónoma.

Algunos sistemas utilizan técnicas que permiten comprender el entorno y tomar decisiones en base a la información que aportan una gran variedad de sensores y dispositivos. Actualmente este enfoque es la base de los sistemas de conducción autónoma más destacados.

Sin embargo, otros sistemas se basan únicamente en el uso de una imagen y la aplicación de técnicas de visión artificial y *Deep Learning*. Existen dos enfoques principales basados en este último esquema:

- El primero recibe típicamente el nombre de *mediated perception* o percepción mediada o intervenida. Se utiliza la imagen de entrada para comprender el entorno utilizando diferentes técnicas, como el reconocimiento de líneas de carril, señales de tráfico… etc., lo cual proporciona una extracción de características explícitas. Esta información permite realizar planificación de rutas y de control. Este es el planteamiento más habitual.
- El segundo enfoque recibe el nombre de *behaviour reflex* o reflejos del comportamiento. Consiste en establecer una correspondencia entre los píxeles de la imagen de entrada directamente a comandos de control, sin necesidad de comprender el entorno de forma explícita. El objetivo es que el sistema aprenda a comportarse a partir de un entrenamiento en el que se graban las imágenes capturadas junto a comandos de conducción. De esta forma el vehículo aprenderá a tomar decisiones únicamente a partir de la imagen.

Aunque el segundo enfoque es más sencillo de realizar, tiene un inconveniente. Desde el punto de vista de la comprensión humana, se desconoce en qué se basa un sistema de este tipo para tomar decisiones. Por ese motivo puede tener más interés el primer enfoque. Emplear extracción de características explícitas, como por ejemplo reconocer el asfalto o las líneas de carril en una imagen, facilita conocer en qué se basa el sistema para tomar las decisiones. Además, permite reducir la dimensionalidad de la entrada del sistema de aprendizaje. Al no utilizar los píxeles de una imagen directamente para tomar decisiones, sino un conjunto de datos o características relevantes, disminuye la complejidad del sistema dado que se reduce la cantidad de parámetros. Para realizar esta extracción de

características suelen utilizarse técnicas de visión artificial, que permiten obtener las coordenadas de las líneas de carril, su inclinación… etc.

El aprendizaje de estos sistemas suele ser supervisado. Los agentes autónomos entrenados de esta forma se basan en la Imitación del Comportamiento (*Behavioral Cloning*). Este método se fundamenta en aprender a realizar una tarea a partir de observaciones e imitarla. Al entrenar un agente autónomo para controlar un vehículo en base al comportamiento desarrollado por un humano, el agente tratará de imitarlo.

También existe la posibilidad de utilizar aprendizaje no supervisado. Esta otra técnica permitiría que el agente desarrolle una capacidad de conducción incluso superior a la habilidad humana. Sin embargo, este tipo de agentes son más difíciles y lentos de entrenar.

El entrenamiento supervisado tiene muchas ventajas. Entre ellas, la posibilidad de recoger sin mucha dificultad grandes cantidades de datos de conducción para entrenar. En cuanto a las situaciones menos comunes o peligrosas, éstas se pueden recrear en simuladores o entornos controlados, de forma que el sistema abarque todas las posibilidades. Además, un sistema cuyo comportamiento imita al de un humano siempre tendrá mejor aceptación por parte de los consumidores. Esto último abre la posibilidad a que un sistema de este tipo pueda ofrecer diferentes estilos de conducción en función de las preferencias de los ocupantes del vehículo.

## 1.2. Objetivos

En este proyecto se desarrollará un vehículo a escala que aprenda a conducir de forma autónoma. Esto implica combinar una serie de elementos importantes. Será necesario un circuito, un vehículo a escala y un ordenador que procese la información recibida por éste y que permita controlarlo de forma supervisada.

El conjunto completo debe permitir realizar una fase de conducción supervisada por el usuario con el objetivo de capturar datos de entrenamiento. La combinación de estos datos con técnicas de visión artificial y redes de neuronas permitirán el aprendizaje de la conducción autónoma.

Tras una fase de entrenamiento en el ordenador vendría una fase posterior que permita al propio vehículo realizar inferencia sobre las imágenes y a conducir de forma autónoma. Esta fase de inferencia se podría realizar tanto desde el ordenador incorporado en el vehículo como desde un ordenador externo con mayor potencia.

A continuación, se describen en detalle los objetivos mencionados.

Se desarrollará un vehículo a escala reducida que disponga de los componentes necesarios para desplazarse y abordar las tareas requeridas. Principalmente estos serán los motores de tracción y de dirección, la alimentación, el ordenador a bordo, una cámara y otros componentes accesorios.

También será necesario el desarrollo de un circuito que reproduzca una carretera delimitada por dos líneas de carril en la que el vehículo pueda circular.

Mediante un ordenador externo y un dispositivo de control vinculado a él (un volante), se desarrollará un sistema que permita el control del vehículo y la obtención de conjuntos de datos de entrenamiento. Este sistema comunicará el ordenador y el vehículo de forma inalámbrica.

Por último, para lograr el desarrollo del agente responsable de la conducción se propone un enfoque dividido en dos tareas.

En primer lugar, se abordará el problema de la extracción de características de las líneas de la carretera a partir de la imagen capturada. En este trabajo se propone usar un tipo de red neuronal llamada *Autoencoder*. Mediante esta técnica se podría obtener un conjunto de características de forma no supervisada. El objetivo es comprobar la viabilidad de esta técnica de *Deep Learning* para desarrollar esta labor.

En segundo lugar, se utilizarán las características obtenidas para entrenar un agente a tomar decisiones de conducción y comprobar si es capaz de recorrer el circuito sin salirse de él. El aprendizaje de la conducción será supervisado, de forma que el agente imite el comportamiento del operario que lo entrene.

Un aspecto importante que se aborda en este trabajo tiene que ver con la creación de datos para entrenamiento. Resulta muy sencillo generar conjuntos inmensos de datos con situaciones de conducción. Pero al aplicar en este caso aprendizaje supervisado, será siempre imprescindible disponer para cada una de las situaciones de las etiquetas necesarias que identifiquen con claridad qué acción se debe tomar en cada caso. Esto se resuelve mediante el entorno de conducción en el que un operario reacciona ante cada una de las situaciones en las que se encuentre el vehículo.

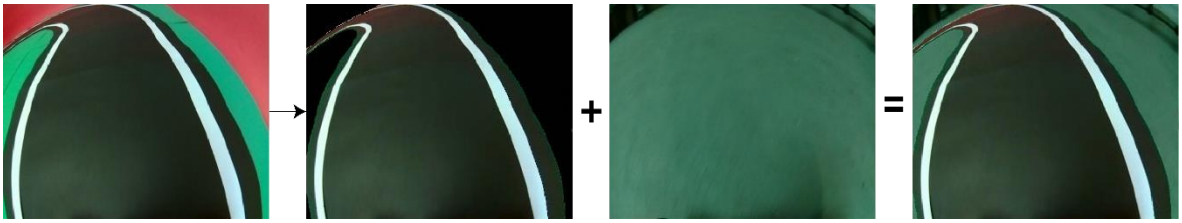
No obstante, aparecen otras tareas en las que la necesidad de etiquetar datos resulta vital. Por ejemplo, la tarea de extraer las líneas de una carretera, deducir su forma y orientación no es tan trivial y su determinación requeriría de un trabajo considerable. Por ello se recurre a los *Autoencoders* que permiten solventar este problema de forma no supervisada.

Otra circunstancia limitante surge cuando se recurre a circuitos reales para entrenar la conducción dado que para realizar la extracción de características (en este caso de las líneas de la carretera) hay que utilizar alguna técnica.

Descartando un proceso manual por demasiado tedioso, quedaría el uso de técnicas de visión artificial. Pero todas ellas podrían producir fallos que empeorarán la calidad del aprendizaje.

Es por ello por lo que se propone la construcción de un escenario auxiliar (basado en chromas) que facilite la extracción de las líneas. Dichas líneas se podrán superponer de forma virtual en imágenes del escenario de conducción.

Estas imágenes aumentadas servirán para entrenar el *Autoencoder*. De esta forma se conseguirá que esta red aprenda a aislar las líneas en escenarios reales, siendo posible realizar el aprendizaje de conducción en ellos.




(a) Imagen capturada en escenario artificial (*chroma)*

(b) Imagen procesada. Carretera aislada.

(c) Imagen capturada en escenario objetivo (conducción)

(d) Imagen aumentada resultante

Figura 1.1: Flujo del proceso de generación de imágenes aumentadas.

### 1.3. Marco Regulador

Se ha dedicado el capítulo 7, MARCO REGULADOR para desarrollar este apartado.

### 1.4. Entorno Socioeconómico

Se ha dedicado el capítulo 8, ENTORNO SOCIOECONÓMICO, para desarrollar este apartado.

### 1.5. Estructura del documento

El contenido presentado en este trabajo se ha dividido en los siguientes capítulos:

- **Introducción**: Presenta la idea general que se va a desarrollar en el proyecto.
- **Estado del arte**: Trata con los métodos y el conocimiento necesario para comprender el trabajo realizado durante el proyecto. Principalmente se estudian aspectos sobre redes de neuronas, tanto históricos como técnicos. También se estudian algunos detalles sobre conducción autónoma, como su origen y la tecnología utilizada en la actualidad. Además, se comentan dos artículos sobre conducción autónoma relevantes para el proyecto.
- **Análisis del problema**: En este capítulo se realiza una descripción del problema que se desea resolver. Para ello también se realiza una evaluación de un conjunto de medios y herramientas que pueden ser utilizados para resolver las tareas. Finalmente, y partiendo de las restricciones y decisiones tomadas en este capítulo, se incluye una especificación de requisitos del proyecto.
- **Diseño y experimentación previa**: El proyecto se ha dividido en cinco fases o agrupaciones de tareas. En este capítulo se describen las cuatro primeras fases, en las que se ha tratado con subproblemas que han permitido abordar las tareas principales del proyecto con mayor facilidad. Sobre todo, se describe la experimentación realizada sobre *Deep Learning* en diferentes dominios y el diseño inicial del vehículo y el sistema de control desarrollado.
- **Diseño y experimentación final**: El contenido de este capítulo corresponde con la última fase del proyecto. Se realiza una descripción detallada del vehículo final utilizado, el entorno de trabajo, la experimentación sobre las tareas principales del proyecto y sus resultados y las conclusiones extraídas.
- **Marco regulador**: En este capítulo se describe la legislación vigente en España sobre vehículos autónomos y las licencias software de los recursos utilizados.
- **Entorno socioeconómico**: En este capítulo se describe el impacto económico y social de la tecnología tratada en este proyecto. También se incluye la planificación y el presupuesto del proyecto.
- **Validación**: En este capítulo se describen las pruebas realizadas para comprobar el cumplimiento de los requisitos del proyecto. Muchas de las pruebas realizadas tienen el objetivo de definir el alcance del trabajo realizado.
- **Conclusiones y trabajos futuros**: Finalmente, en este capítulo se describen las conclusiones extraídas sobre todo el trabajo realizado en el proyecto. También se incluyen trabajos futuros con los que se podría continuar con el desarrollo de este proyecto.

## 1.6. Glosario

A lo largo de esta memoria se han utilizado algunas abreviaturas y acrónimos. A continuación, se indica el significado de todos ellos.

| | |
|---|---|
| AE | Autoencoder |
| API | Application Programming Interface |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| DGT | Dirección General de Tráfico |
| DNN | Deep Neural Network |
| FPS | Frames Per Second |
| GPS | Global Positioning System |
| GPU | Graphics Processing Unit |
| Grad-CAM | Gradient-weighted Class Activation Mapping |
| HSV | Hue Saturation Value |
| HTTP | Hypertext Transfer Protocol |
| HUD | Head-Up Display |
| I2C | Inter-Integrated Circuit |
| JPEG | Joint Photographic Experts Group |
| LIDAR | Laser Imaging Detection and Ranging |
| MLP | Multilayer Perceptron |
| MSE | Mean Squared Error |
| NTCS | National Television System Committee |
| PAL | Phase Alternating Line |
| PVP | Precio de Venta al Público |
| PWM | Pulse-Width Modulation |
| ReLU | Rectified Linear Unit |
| RGB | Red Green Blue |
| RPM | Revolución Por Minuto |
| SAE | Society of Automobile Engineers |
| SBC | Single Board Computer |
| SDAE | Stacked Denoising Autoencoder |
| TCP | Transmission Control Protocol |
| TFG | Trabajo de Fin de Grado |
| UDP | User Datagram Protocol |

# 2. ESTADO DEL ARTE

En este capítulo se describirán los métodos y el conocimiento previo necesario para comprender el trabajo realizado. En concreto, se tratará principalmente sobre redes de neuronas, tanto de aspectos históricos como técnicos. También se abordará la conducción autónoma en la actualidad y se describirán artículos relevantes para el proyecto.

## 2.1. Redes de Neuronas Artificiales

### 2.1.1. Breve historia sobre las redes de neuronas

En el año 1943, dos investigadores del MIT, Warren McCulloch y Walter Pitts, definieron el primer modelo computacional de una red neuronal [5]. Fueron capaces de intuir una forma alternativa de resolver problemas, utilizando dispositivos formados por una red de unidades interconectadas llamadas neuronas por su inspiración biológica. No desarrollaron ningún algoritmo de aprendizaje, pero su artículo serviría de base para el desarrollo posterior de las redes de neuronas artificiales.

Donald Hebb en 1949 hizo una publicación [6] en la que propuso una teoría sencilla del aprendizaje de las neuronas biológicas. Hebb planteó la hipótesis de que ese aprendizaje se encuentra en la creación y fortalecimiento de las conexiones entre unidades neuronales, las uniones sinápticas.

El modelo definido por McCulloch y Pitts consiste en una serie de neuronas binarias con umbral. En cada una de ellas se calcula la suma ponderada de las entradas. Esta suma está formada por el producto de cada entrada de la neurona por un peso sináptico. En la salida se puede generar un pulso si la suma ponderada supera un umbral preestablecido.

$$z = \sum_{i}^{N} w_i \cdot x_i \tag{2.1}$$

*N = número de entradas de la neurona,*

*w = peso sináptico,*

*x = valor de la entrada.*

$$y = \begin{cases} 1 & \text{si } z \geq \theta \\ 0 & \text{en otro caso} \end{cases} \tag{2.2}$$

*$\theta$ = umbral preestablecido,*

*y = salida de la neurona.*

El ajuste de los pesos sinápticos y umbrales tenía que realizarse de forma manual, hasta que Clark, Farley, Minsky, Rosenblatt y Uttley comenzaron a ajustar de forma dinámica estos pesos y dotaron a las redes neuronales la capacidad de aprender. En 1958, Rosenblatt presentó el perceptrón simple [7], el modelo más popular de los años 60.

El perceptrón presentaba un inconveniente, solo era capaz de resolver problemas de tipo lineal. Marvin Minsky y Seymour Papert publicaron en 1969 un libro en el que describían las limitaciones del perceptrón [8]. Este fue un momento crucial ya que fue uno de los desencadenantes de la entrada en un periodo gris conocido como invierno de la Inteligencia Artificial.

Más de diez años después, las redes neuronales resurgieron con fuerza con la llegada de un algoritmo llamado *backpropagation*, propagación de errores hacia atrás. Este algoritmo combinado con una técnica de optimización como el descenso del gradiente,

permitía la posibilidad de entrenar redes de neuronas multicapa, tanto para problemas lineales como no lineales. Esta técnica se popularizó tras la publicación de un artículo de David Rumelhart, Geoffrey Hinton y Ronald Williams en 1986 [9], aunque también se atribuye el descubrimiento a David Parker [10] y Yann LeCun [11].

Con el cambio de siglo las redes de neuronas perdieron algo de popularidad a favor de otras técnicas, como las máquinas de vectores de soporte (SVM). Entre los años 2006 y 2012 las redes de neuronas vuelven a tener popularidad bajo el nombre de aprendizaje profundo, D*eep Learning*. Esto ocurrió gracias a la mejora en la potencia de computación del hardware (con el uso de GPU) y la gran cantidad de datos disponibles para entrenar.

En 2006, la publicación de tres artículos jugó un papel decisivo en la reaparición de las redes neuronales. Los autores de estos trabajos fueron Geoffrey Hinton [12], Yoshua Bengio [13] y Yann LeCun [14]. La importancia de estos trabajos como fundamentales del llamado *Deep Learning* fue reconocida con el premio Alan Turing 2018 [15].

Poco tiempo después aparecieron algoritmos que se comenzaron a usar en una amplia variedad de problemas de aprendizaje. Uno de los ejemplos más conocidos es el de la red denominada AlexNet [16]. Esta red, creada por Alex Krizhevsky en 2012, consiguió muy buenos resultados en una competición de reconocimiento de objetos.

Las técnicas de *Deep Learning* pertenecen a un subconjunto del aprendizaje automático, que realizan ingeniería de características de forma automática. Las redes de neuronas de este nuevo campo están formadas por múltiples capas en las que se obtienen características cada vez más complejas en función de la profundidad. Han mostrado ser muy útiles para el análisis de datos no estructurados como audio o imágenes.

### 2.1.2. Deep Neural Networks

Las redes de neuronas profundas son los modelos más importantes del *Deep Learning*. Están caracterizadas por tener múltiples capas ocultas entre la capa de entrada y la capa de salida. Este tipo de redes puede resolver tanto problemas lineales como no lineales. Además, forman la base de muchas aplicaciones comerciales, así como las redes convolucionales.

Uno de los casos especiales de este tipo de redes son las D*eep Feedforward Networks*. El nombre de *feedforward* se debe a que en estas redes la información fluye siempre hacia delante. Existe otro tipo en el que sí se permite la circulación de información hacía atrás, estas son las redes de neuronas recurrentes.

Al diseñar las redes de neuronas profundas es necesario definir la arquitectura: el número de capas ocultas, cómo van a estar conectadas las distintas capas y el número de neuronas por capa. Además, la aparición de las capas ocultas implica definir la función de activación a utilizar en ellas (funciones como SoftMax, sigmoide, ReLU, lineal…).

Los modelos lineales son capaces de ajustarse únicamente a funciones lineales. Sin embargo, la mayoría de los sistemas necesitan tratar con funciones no lineales. El teorema de aproximación universal [17] expone que una *feedforward network* con salida lineal, con al menos una capa oculta y con el número suficiente de neuronas puede aproximar cualquier función. Es necesario que las neuronas de la capa oculta utilicen una función de activación no lineal (específicamente, las funciones denominadas *squashing* en el campo de las redes de neuronas).

Aunque un modelo con una sola capa oculta podría aproximar cualquier función, utilizar modelos profundos puede aportar beneficios. Se cree que usar varias capas ocultas

facilita aproximar la función mediante una composición de funciones más simples y mejora la capacidad de generalización del modelo [18].

### 2.1.3. Redes de neuronas convolucionales

Las redes convolucionales o convolutivas (CNN) son las redes de neuronas artificiales que habitualmente se utilizan para resolver problemas que requieren procesar imágenes. Este tipo de red saca provecho a las entradas estructuradas. Esto significa que mientras que podemos desconocer la relación entre las distintas entradas de un perceptrón multicapa, en una red convolucional se aprovecha la estructura de las entradas. Una entrada vectorial (1D) podría ser sonido, una matriz (2D) podría ser una imagen, y un tensor (>2D) podría ser una imagen a color, un vídeo… etc.

La salida de una red de este tipo puede ser no estructurada (salida convencional, para problemas de clasificación o regresión) o puede ser estructurada (un sonido, imagen o vídeo).

Estas redes se caracterizan por utilizar dos operaciones, la de convolución (en la capa convolucional) y la de reducción o *pooling* (en la capa de *pooling*).

La capa convolucional permite detectar características o patrones. Una capa densa sería capaz de detectar patrones en una imagen, pero lo haría en el espacio global de esta. Sin embargo, una capa convolucional detecta patrones en una ventana bidimensional dentro de la imagen. De esta forma, se obtiene un filtro capaz de detectar un patrón en cualquier parte de la imagen.

Las capas convolucionales están definidas por dos parámetros. En primer lugar, el tamaño de la ventana o *kernel*. Esta ventana se desplaza por toda la imagen realizando la operación. En segundo lugar, el número de filtros de la capa convolucional. Los filtros son los encargados de obtener características de la imagen. A medida que se avanza por las capas de la red, las características obtenidas son más complejas. Se puede observar una representación gráfica de la obtención de características realizada por esta capa en la Figura 2.1.

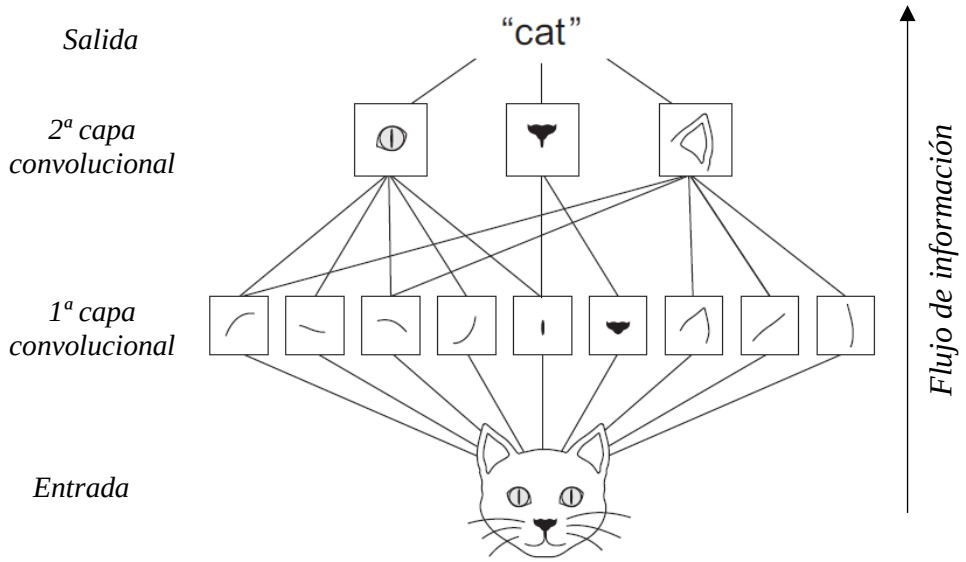



Figura 2.1: Representación de la extracción de características en CNN.

*Los primeros filtros de la red extraen características simples, como bordes o líneas. Los siguientes filtros toman las características simples para extraer otras más complejas (un ojo, una oreja…). Imagen adaptada de [19].*

La capa de *pooling* suele ser aplicada inmediatamente después de las capas convolucionales. El objetivo de esta capa es realizar una simplificación de la información dada por la capa convolucional, es decir, reducir el tamaño para que las capas posteriores trabajen con una dimensionalidad menor. Existen varias formas de aplicar esta operación, aunque la más conocida recibe el nombre de Max-Pooling. En la Figura 2.2 se puede observar un ejemplo de su funcionamiento. Usando una ventana de 2x2 se consigue reducir a la mitad el tamaño de la imagen.

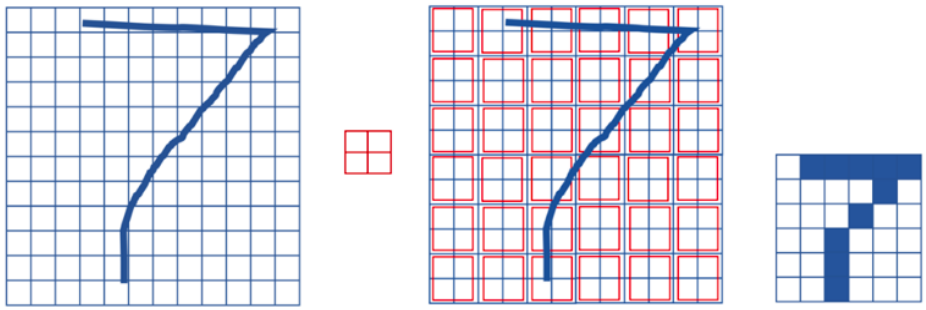


Figura 2.2: Representación gráfica de la operación de *Max-Pooling.*

*La operación de Max-Pooling toma el valor máximo (en este caso, existencia o no de trazo) en la ventana de pooling. Este ejemplo usa una ventana de 2x2. Fuente: [20]*

En las redes convolucionales se aplican sucesivas veces este par de operaciones, consiguiendo en cada aplicación una detección de patrones cada vez más complejos. En las primeras capas se pueden detectar elementos simples (bordes, líneas rectas…), pero en capas más profundas se pueden detectar elementos más complejos (ojos, orejas…, en caso de usar imágenes de rostros). Conectando la salida de estas sucesivas operaciones a la entrada de una red densa, como un MLP, se podría llegar a una conclusión a partir de las características detectadas, por ejemplo, la clasificación de un número manuscrito (MNIST [21]). Este ejemplo se puede visualizar en la Figura 2.3.

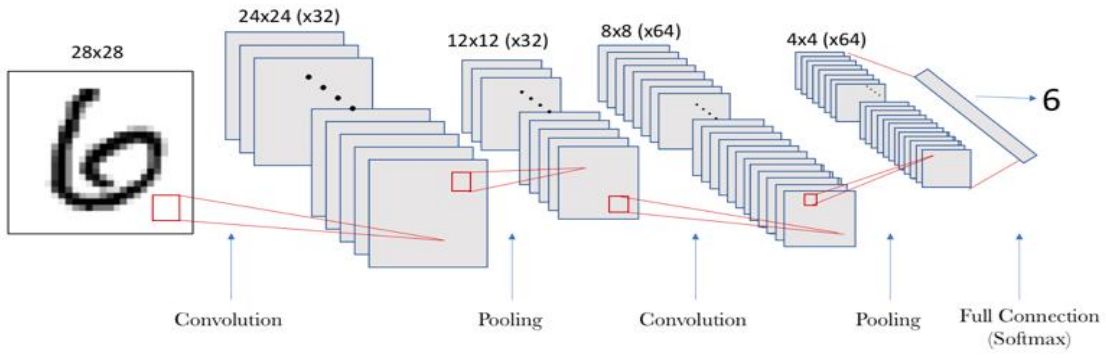



Figura 2.3: Representación de una red convolucional.

*Esta figura representa un ejemplo del funcionamiento de una CNN para un dígito del MNIST. Las capas de convolución y pooling se encargan de la extracción de características. La parte de Full Connection hace referencia a una red densa que clasifica entre 10 clases. Fuente: [20]*

### 2.1.4. Regularización – Introducción de ruido

Durante el proceso de aprendizaje se pueden aplicar múltiples técnicas para mejorar la red obtenida. Estas técnicas reciben el nombre de regularizadores y principalmente están destinados a mejorar el error de generalización de la red.

Una de estas técnicas es la introducción de ruido pseudoaleatorio durante el proceso de entrenamiento de la red. El efecto de añadir ruido es similar a ampliar el conjunto de datos de entrenamiento. La ampliación del conjunto de datos de entrenamiento realizada mediante la técnica de *data augmentation* (traslaciones, rotaciones… etc.) mejora la robustez de la red frente a determinadas transformaciones de los datos de entrada. Sin embargo, la ampliación del conjunto mediante la introducción de ruido mejora de forma genérica la robustez de la red.

La introducción de ruido se puede realizar de distintas formas. El ruido puede añadirse sobre los datos de entrada, ampliando el conjunto de entrenamiento. Aunque también puede actuar sobre los niveles de activación de las capas ocultas o sobre los pesos sinápticos de la red.

En primer lugar, si se realiza sobre los datos de entrada, se modifican los casos de entrenamiento añadiendo una cantidad de ruido en ellos. De esta forma, se consigue ampliar el conjunto de entrenamiento y obtener una red más robusta. Esta será capaz de tratar con datos con ruido y también habrá mejorado su capacidad de generalización sobre datos sin ruido. El efecto también se puede relacionar con que al entrenar la red con ruido, más neuronas de la red contribuyen de manera efectiva en el funcionamiento de esta [22]. Sin embargo, si no se utiliza ruido en las entradas durante el entrenamiento, es más probable que algunas neuronas sean irrelevantes (*dead units)*.

El ruido puede ser añadido de forma aditiva o multiplicativa. De forma aditiva y utilizando imágenes, consiste en sumar una pequeña cantidad positiva o negativa a cada píxel de la imagen. La cantidad aplicada viene dada por una distribución, que suele ser una gaussiana. Dependiendo de los parámetros de la distribución (media y desviación típica), la cantidad de ruido aplicado es diferente, se puede ver un ejemplo en la Figura 2.4.

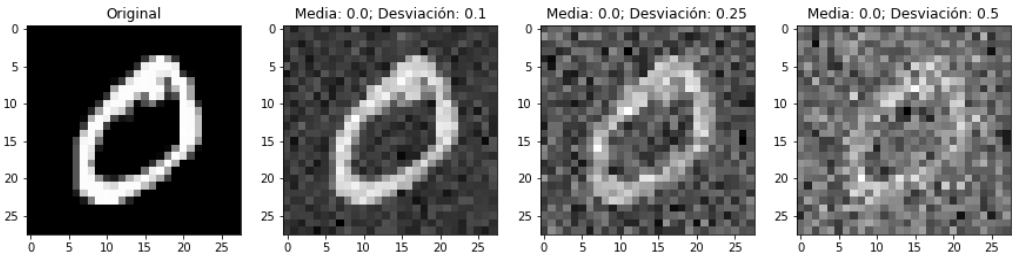



Figura 2.4: Introducción de ruido en una imagen de MNIST.

*Utilizando una distribución gaussiana, se introduce ruido sobre una imagen de MNIST. Se muestran tres ejemplos en los que aumenta la desviación de la distribución y por tanto aumenta la cantidad de ruido añadida. El valor de cada píxel se mantiene entre 0 y 1,0 tras introducir el ruido.*

A pesar de que el ruido sobre las entradas sea la estrategia más frecuente, este se puede aplicar sobre otros elementos de la red neuronal. Este es el caso de la adición de ruido sobre los pesos, conocido como *weight noise* o *synaptic noise*, empleado principalmente sobre redes recurrentes.

Además, otra de las técnicas más frecuentemente utilizadas es la adición de ruido sobre las capas ocultas. En este caso el ruido actúa directamente sobre las características que se extraen de los datos de entrada. Una estrategia muy parecida a esta y muy utilizada en *Deep Learning* es el uso de la regularización *Dropout*. El funcionamiento consiste en desactivar las neuronas de la red con una probabilidad, de forma que estas aprendan características sin depender siempre de la salida de otras neuronas, es decir, evita correlaciones espurias.

### *2.1.5. Autoencoders*

Un *Autoencoder* (AE) o autocodificador es un tipo de red de neuronas cuyo objetivo es copiar los datos de entrada a la salida. Tiene una capa oculta que describe una representación de la entrada, esta recibe el nombre de vector latente. Esta red está dividida en dos partes, la de codificación y la de decodificación.

En primer lugar, en la parte de codificación o *Encoder* se realiza la codificación de los datos de entrada, *x*, en el vector latente, *z*. Este proceso se realiza mediante una función de transformación *f:* $f(x) = z$. En segundo lugar, en la parte de decodificación o *Decoder* se recupera la entrada, ahora $\tilde{x}$, partiendo del vector latente. El objetivo es reconstruir lo mejor posible la entrada *x*. Este segundo proceso se realiza mediante una función de transformación *g:* $g(z) = \tilde{x}$. Se puede ver la estructura en bloques en la Figura 2.5.

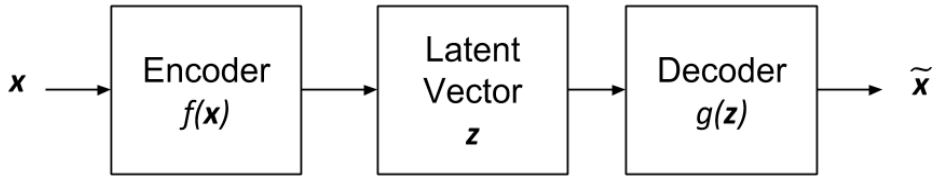



Figura 2.5*:* Diagrama en bloques de un *Autoencoder.*

*Se realiza un mapeo de la entrada x a la salida, llamada reconstrucción, x̃, utilizando una representación interna, el vector latente, z. Fuente: [23]*

Los *Autoencoders* normalmente se componen de una capa para la codificación, una capa oculta y otra capa de decodificación. Sin embargo, no es un requisito realizarlo de esa forma. Utilizar más de una capa para la codificación y decodificación (*Stacked Autoencoder* o *Deep Autoencoder*) puede tener muchas ventajas.

Como ya se ha hablado en la sección 2.1.2 Deep Neural Networks, utilizar redes con profundidad tiene beneficios. Como los *Autoencoders* son redes del tipo *feedforward*, estas ventajas también se pueden aplicar a ellos.

La adición de capas intermedias facilita la reducción de la dimensionalidad progresiva en el caso de la codificación, y una reconstrucción progresiva en el caso de la decodificación. Experimentalmente, los *Stacked Autoencoders* consiguen mejores representaciones de los datos en el vector latente.

#### 2.1.5.1. Undercomplete Autoencoders

En los autocodificadores incompletos o *Undercomplete Autoencoders*, la dimensión del vector latente (z) es inferior al tamaño de la entrada. Este es el tipo de *Autoencoder* más común, ya que ofrece diversas aplicaciones y beneficios. Una de las aplicaciones más comunes de este tipo de *Autoencoders* consiste en la reducción de la dimensionalidad de entrada.

La dimensión del vector latente normalmente es mucho menor que la dimensión de la entrada. Esto provoca que en el vector latente solo se mantengan aquellas características que diferencian una entrada del resto. La tarea que se está realizando al fin y al cabo es una extracción automática de características.

Si la dimensión del vector latente fuese superior (*Overcomplete Autoencoders*) o igual (*Regularized Autoencoders*) al tamaño de entrada, la red solo tendría que copiar la entrada en el vector, aunque existen técnicas para que estos últimos casos sean útiles.

Para entrenar este tipo de redes se utiliza, al igual que en el resto, una medida de error. En este caso, una función de error se encargará de medir como de similares son la entrada, *x*, y la salida obtenida, $\tilde{x}$. Durante el entrenamiento, el *Autoencoder* trata de minimizar esta función de error, consiguiendo así la mayor similitud entre *x* y $\tilde{x}$. En la ecuación (2.3) se puede observar un ejemplo de una posible función de error usando el Error Cuadrático Medio (MSE):

$$S(x, \tilde{x}) \; = \; MSE\;(x, \tilde{x}) = \frac{1}{m}\sum_{i=1}^{i=m}(x_i - \tilde{x}_i)^2 \qquad (2.3)$$

*m = dimensión de salida, por ejemplo, en el caso de una imagen: m = alto x ancho x canales,*

*$x$ = entrada,*

*$\tilde{x}$ = reconstrucción de la entrada.*

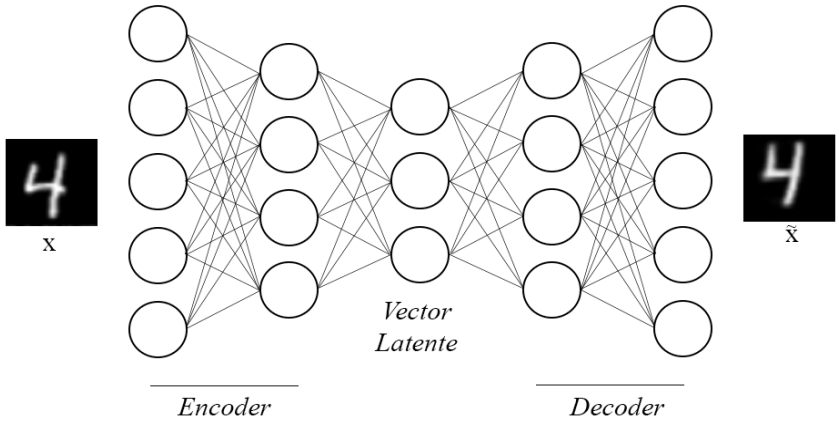



Figura 2.6: Arquitectura de un S*tacked Undercomplete Autoencoder*.

*Dado que la dimensión del vector latente es menor que la de la entrada, la red aprende a extraer características de los datos. Estas características permiten realizar una reconstrucción de la entrada.*

### 2.1.5.2. Denoising Autoencoders

Este tipo de *Autoencoders* puede servir para eliminar el ruido de los datos de entrada. Uno de los efectos colaterales de eliminar este ruido es conseguir una mejor representación de los datos en el vector latente (mejor extracción de características) [24]. El objetivo final de este tipo de redes suele ser obtener una buena representación de los datos de entrada.

A diferencia de los *Autoencoders* convencionales, el entrenamiento de este tipo de redes es algo diferente. En este caso, en la entrada se introducen los datos originales con ruido y como salida esperada se introducen los datos originales sin ruido. Tras ese entrenamiento se espera que, al introducir un dato con ruido, la red elimine dicho ruido. En la Figura 2.7 se presentan ejemplos de datos de este tipo de red usando el conjunto de datos del MNIST.

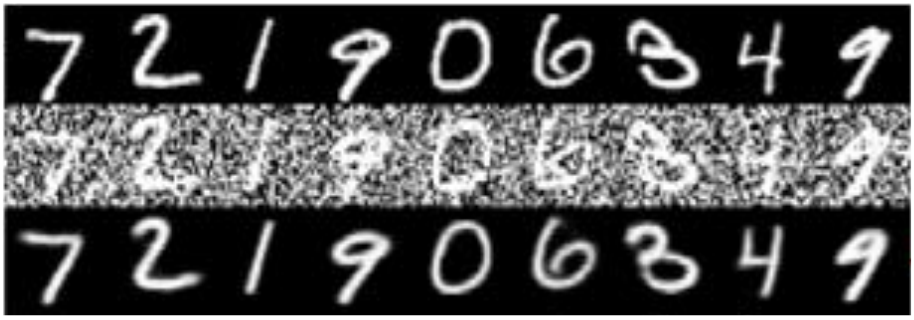


Figura 2.7: Ejemplo de MNIST en *Denoising Autoencoders.*

*Imágenes originales de MNIST (fila superior); imágenes con ruido, entrada de la red (fila de en medio); imágenes sin ruido, reconstrucción dada por la red (última fila). Imagen adaptada [23].*

Al entrenar la red aplicando ruido sobre los datos de entrada, se obtiene una red más robusta y con más capacidad de generalización. Puede ser muy útil ya que permite reconstruir los datos de entrada cuando estos se encuentran en alguna situación más complicada.

Otra de las diferencias reside en la función de error. En este caso se mide la similitud entre los datos originales (sin ruido) y los obtenidos. En la siguiente ecuación se puede observar un ejemplo de una posible función de error usando el Error Cuadrático Medio (MSE):

$$S(x_{Orig}, \tilde{x}) \; = \; MSE\;(x_{Orig}, \tilde{x}) = \frac{1}{m}\sum_{i=1}^{i=m}(x_{Orig_i} - \tilde{x}_i)^2 \qquad (2.4)$$

*m = dimensión de salida, por ejemplo, en el caso de una imagen: m = pixeles = alto x ancho x canales,*

$x_{Orig}$ *= dato original de entrada (sin ruido),*

$\tilde{x}$ *= entrada reconstruida.*

### 2.1.5.3. Otros tipos de *Autoencoders*

Teniendo en cuenta los tipos autocodificadores vistos anteriormente, se pueden realizar combinaciones de ellos dependiendo del objetivo y aprovechar sus ventajas. A continuación se presentan algunas variantes o aplicaciones de *Autoencoders* que se utilizan en el proyecto:

- ***Stacked Denoising Autoencoder (SDAE)***: Este tipo de red combina el S*tacked Autoencoder* y el *Denoising Autoencoder*, consiguiendo un autocodificador que aprovecha las ventajas de ambos.
- ***Convolutional Autoencoder (CAE)***: Este tipo de red utiliza capas convolucionales. Cuando se utilizan imágenes como entrada de la red, cobra sentido utilizar este tipo de capa.
- ***Autoencoders*** aplicados a la **segmentación de imágenes**: Los autocodificadores también se pueden utilizar para realizar tareas de segmentación. En este tipo de red se introduce en la entrada una imagen y en la salida esperada se introduce la imagen segmentada. Realizando el entrenamiento se podría obtener una red que aprenda a realizar la segmentación [25].

  Una tarea parecida a esta es la **extracción de una zona de interés** de la imagen de entrada. Introduciendo en la salida esperada la imagen de entrada pero

únicamente manteniendo aquello que interesa, se podría obtener una red que aprenda a aislar elementos de interés en una imagen.

## 2.2. Conducción autónoma

Desde los años 1920 se han realizado experimentos sobre conducción autónoma. Fue en los años 1950 cuando comenzaron a aparecer pruebas prometedoras que provocaron interés por trabajar en ello.

Los primeros vehículos con funciones de conducción autónoma, como el seguimiento de carretera, aparecieron en los años 1980. En 1984, la Universidad Carnegie Mellon con un proyecto llamado ALVINN [26] y en 1987 varios científicos de universidades europeas con el proyecto EUREKA PROMETHEUS [27].

Desde entonces, han sido muchas las compañías y organizaciones que investigan el desarrollo de vehículos autónomos.

### 2.2.1. Técnicas utilizadas actualmente

Actualmente se utiliza una gran variedad de tecnologías en los vehículos autónomos que aportan información sensorial sobre el entorno. Los sistemas de control avanzados compuestos por estas tecnologías son capaces de interpretar la información para formar rutas de navegación y detectar obstáculos y señales. Estos sistemas están compuestos por:

- Dispositivo LIDAR: Es un láser que mide distancias entre los objetos del entorno mediante un haz de luz. A partir de dichas distancias se pueden construir mapas 3D del entorno mediante los que detectar obstáculos y objetos.
- Radares: Utilizan ondas electromagnéticas para detectar objetos alrededor del vehículo. Los datos obtenidos suelen ser combinados con los del LIDAR.
- Sensores ultrasónicos: Son sensores muy comunes en todo tipo de vehículos para aportar ayuda principalmente en el proceso de aparcar.
- GPS: Es el sistema basado en navegación por satélite. Puede ser utilizado para obtener la posición actual exacta junto con otras tecnologías, ya que por sí mismo no puede ofrecer información muy precisa.
- Unidad de medición inercial: Incorpora sensores como acelerómetros, giroscopios y magnetómetros para aportar información más precisa sobre la posición.
- Cámaras: Aportan imágenes de la carretera o del entorno situado en frente del vehículo. Estas imágenes pueden ser tratadas mediante técnicas de *Deep Learning* o procesamiento de imagen para realizar diferentes tareas dentro de la conducción autónoma.

Los datos obtenidos por las herramientas mencionadas se recogen en un ordenador que los procesa y se encarga de tomar las decisiones necesarias.

Con la aparición del *Deep Learning* se abrieron muchas posibilidades en este campo utilizando imágenes. Con tareas como la identificación o clasificación de imágenes, la detección y reconocimiento de objetos y la segmentación de escenas. Estos métodos combinados con las tecnologías anteriores están comenzando a usarse para comprender el entorno. Sin embargo, la investigación en redes del tipo CNN ha proporcionado métodos de aprendizaje del control del vehículo basados únicamente en la imagen de entrada.

### 2.2.2. *End to End Learning for Self-Driving Cars*

*Bojarski* et al. [28] propusieron un método de aprendizaje del control de un vehículo basado únicamente en la imagen de entrada y una red de neuronas convolucional. Las imágenes tomadas en este trabajo se obtienen desde una cámara frontal colocada en un vehículo.

La red de neuronas realiza una correspondencia de los pixeles de imágenes de conducción directamente a comandos de dirección. Este método tiene la ventaja de que el sistema de conducción se simplifica ya que es la red convolucional la que aprende automáticamente a entender el entorno y la planificación.

Con este sistema pudieron demostrar lo realmente potente que es este método de aprendizaje. Sin necesidad de realizar procesamientos previos, la red era capaz de aprender las características necesarias de una carretera, con líneas o sin ellas, tan solo aportando como etiqueta el control de dirección dado por el humano.

Comparándolo con otros métodos que realizan una descomposición explícita del problema, como la detección de líneas o planificación de ruta, consiguieron un sistema que optimiza todos los pasos de procesamiento simultáneamente y de forma automática. Además, argumentan que este sistema tiene un mayor rendimiento. El uso de criterios como la detección de líneas facilita la interpretación humana, pero este método no garantiza el máximo rendimiento.

Con este trabajo consiguieron demostrar empíricamente como una CNN es capaz de aprender la tarea completa de seguir una carretera o un camino sin aplicar técnicas de comprensión explícita del entorno.

### 2.2.3. *Deep Learning Technique-Based Steering of Autonomous Car*

A diferencia del artículo anterior, en este [29] no se utiliza una red de neuronas convolucional. En este artículo *Y. Yang* et al. proponen un *Stacked Autoencoder* cuyo objetivo es aprender las características necesarias de la conducción para, finalmente, conseguir el giro autónomo del vehículo.

El trabajo propuesto consiste en un vehículo a escala reducida formado por una Raspberry Pi que controla el vehículo y toma imágenes mediante una cámara montada en la parte superior.

El entrenamiento se realiza en dos fases. En la primera fase se hace un pre-entrenamiento de un S*tacked Autoencoder.* En este entrenamiento la red aprende a extraer las características del circuito en el que conducirá.

En la segunda fase se realiza un proceso de *fine-tuning.* Se toma el *Stacked Autoencoder* y se le añade una capa de salida de tres neuronas. En esta segunda fase se entrena toda la red pero en este caso se realiza con las etiquetas de giro. El entrenamiento se realiza con 3 clases, giro a izquierda, recto o giro a derecha.

Una de las mejoras que añaden en su trabajo es la adición de una capa de *Batch Normalization* para ajustar la distribución de los datos de entrada y además tener el efecto que proporciona como regulizador.

En cuanto a los resultados obtenidos, llegan a la conclusión de que el *Autoencoder* propuesto (con el uso de preentrenamiento y *Batch Normalization)* tiene una convergencia más rápida y es más robusto que un *Autoencoder* tradicional. Experimentalmente pueden observar como el vehículo es capaz de recorrer el circuito

incluso realizando cambios en la escena, lo que demuestra que la red consigue generalizar o fijarse en características propias del circuito y no del entorno en sí.

Con relación al trabajo realizado en este artículo, en este proyecto se aportan ciertas contribuciones. En particular, se utilizará un escenario artificial basado en chromas que permitirá diferenciar las líneas de la carretera del resto de elementos de la escena, por lo que se extraerán únicamente las características que corresponden a dichas líneas.

Además, este escenario permitirá la generación de imágenes aumentadas, que permite un entrenamiento virtual de la red en escenarios realistas.

Otra de las contribuciones principales es el uso de un *Autoencoder* más elaborado, en concreto, un *Denoising Stacked Convolutional Autoencoder*, que mediante la introducción de ruido en los datos de entrada (*Denoising*) ha permitido mejorar la robustez del modelo creado.

## 2.3. Espacios de Color

Puesto que en este proyecto también se ha tratado con aspectos relacionados con el color, a continuación se describen las características de los principales espacios de color [30]. Será necesario seleccionar un modelo de color adecuado para identificar correctamente una zona de interés en una imagen en base a la similitud de colores.

El color puede ser representado digitalmente por diferentes modelos. El más conocido y utilizado por los monitores de vídeo es el modelo de color RGB. A pesar de ello, existen otros modelos que pueden ofrecer distintas ventajas, por ejemplo, proporcionar un manejo de color mucho más intuitivo, o ser más eficientes para su uso con hardware.

### 2.3.1. RGB

Este modelo está basado en la percepción humana del color, el uso de tres pigmentos visuales en los conos de la retina. Estos tres pigmentos son el rojo, verde y azul. Comparando las intensidades de una fuente luminosa, se puede percibir el color de la luz.

Esta teoría es la base para la visualización de colores en monitores de vídeo debido a que su transformación es inmediata. Sin embargo, el uso de este modelo para usuarios es complejo ya que no es sencillo utilizarlo para diferenciar colores.

### 2.3.2. YIQ, YUV, YCrCb

Los monitores RGB necesitan señales separadas para las tres componentes. Sin embargo, los monitores de televisión utilizan una señal compuesta por lo que utilizan otro modelo de color. La codificación de color de televisión NTSC se denomina modelo YIQ, en el caso de PAL, recibe el nombre de YUV. Otro modelo parecido a estos es el utilizado en la codificación de vídeo digital JPEG, denominado YCrCb.

Los parámetros de este modelo son la luminancia o el brillo (Y) y dos componentes cromáticas, tono y pureza (I y Q). Para televisores en blanco y negro tan solo es necesario la componente Y. Las componentes I (rango de naranja a azul) y Q (rango de púrpura a verde) se encargan de añadir el aspecto cromático a la imagen.

Este modelo de color también está orientado a hardware, por lo que no es sencillo utilizarlo para diferenciar o seleccionar colores.

### 2.3.3. HSV

Este es el modelo más utilizado para realizar selecciones de color en una interfaz ya que está basado en conceptos intuitivos. Mediante este modelo se puede especificar un

color seleccionando un color base, o color espectral y la cantidad de blanco y negro que hay que añadir para obtener las diferentes sombras, tintas y tonos.

Los parámetros de este modelo son el tono (H), saturación (S), y valor (V). Este modelo puede ser representado geométricamente por un cono con forma hexagonal. El contorno del hexágono representa los distintos tonos, la saturación se mide según el eje horizontal (dirección radial del cono) y el valor por el eje vertical (eje central del cono).

El tono (H) queda representado por cada vértice del hexágono, los colores complementarios están separados por 180º. La saturación (S) es el parámetro que determina la pureza de un color, en el centro del hexágono se encuentra el blanco. Por último, el valor (V) varía entre el vértice del cono y el plano, el vértice representa el negro y mientras aumenta la cantidad de este parámetro, se recorre la escala de grises. La gran ventaja de este modelo es la posibilidad de comparar colores utilizando únicamente el valor de tono (H).

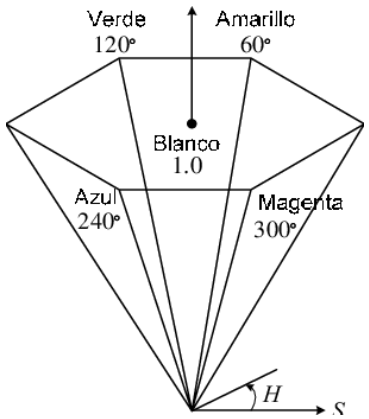



Figura 2.8: Representación del espacio de color HSV.

# 3. ANÁLISIS DEL PROBLEMA

En este capítulo se realiza una descripción del problema que se quiere resolver. También se muestra una evaluación de diferentes métodos, medios y herramientas entre las que se ha tenido que decidir para resolver las tareas del problema. Finalmente, se incluye la especificación de requisitos del proyecto.

El desarrollo del proyecto se basa en un ciclo de vida en espiral. Un análisis inicial podría no determinar todos los elementos necesarios para resolver las tareas, dado que muchas decisiones dependerán de los resultados que se obtengan en fases experimentales. El proyecto se ha descompuesto en análisis inicial, diseño por fases y diseño final.

## 3.1. Descripción del problema

El objetivo fundamental del proyecto consiste en el desarrollo de un vehículo a escala reducida que aprenda a seguir un recorrido delimitado por dos líneas de carretera. Además, será necesario el desarrollo de un sistema que permita el control supervisado y la obtención de datos para entrenar. El agente autónomo desarrollado se encargará de obtener una serie de características de dichas líneas a partir de las que aprenderá a realizar la conducción.

El problema mencionado se dividirá en dos subproblemas. El primero de ellos consiste en la tarea de extracción de características de las líneas de la carretera y el segundo en el aprendizaje de la conducción.

La entrada al sistema de conducción autónoma será una imagen. Esta corresponderá con lo que el vehículo visualiza desde su parte frontal. El sistema debe extraer características de las líneas de la carretera presentes en la imagen y utilizarlas para aprender a realizar la conducción. El aprendizaje del control del vehículo estará basado en el uso de redes de neuronas artificiales mediante aprendizaje supervisado. La salida del sistema de conducción debe corresponder con un valor relacionado con la dirección del vehículo. Se puede observar un diagrama de este sistema en la Figura 3.1.

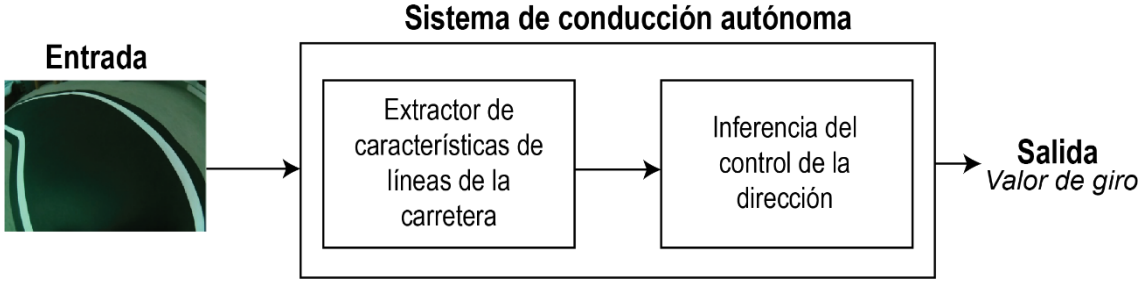



Figura 3.1: Diagrama del sistema de conducción autónoma.

*La entrada del sistema es la imagen capturada por el vehículo. Se deben extraer características relacionadas con las líneas de la carretera y utilizarlas para inferir el control de la dirección del vehículo. La salida del sistema corresponde con un valor relacionado con la dirección.*

Se desarrollará un vehículo a escala reducida que permita abordar las tareas del proyecto. Para ello se estudiarán componentes como el ordenador a bordo, otros relacionados con mecánica y electrónica para realizar el movimiento y con la captura de las imágenes. El procesamiento necesario se realizará en un SBC (*Single Board Computer*), es decir, en un computador completo en un solo circuito. A excepción del entrenamiento de redes de neuronas, se deben realizar, preferiblemente, todas las tareas de procesamiento en el vehículo.

Ya que se desea desarrollar una conducción realista, la dirección será controlada por las ruedas delanteras mediante un eje de dirección. El empuje del vehículo lo desarrollarán dos motores en las ruedas traseras.

Otro de los objetivos del proyecto consiste en el desarrollo de un sistema de control y obtención de datos de entrenamiento.

Este sistema debe ofrecer el control supervisado del vehículo de forma remota. Se estudiarán dispositivos de control que permitan realizar la conducción de forma realista. El operario que controle el vehículo debe visualizar las imágenes capturadas por el vehículo en tiempo real, como si estuviese montado en el vehículo.

Este sistema también se encargará de formar conjuntos de datos de entrenamiento compuestos por la imagen de conducción, un valor que indique el estado de la dirección y, opcionalmente, un valor que indique el estado de la velocidad.

El valor relacionado con el giro del vehículo, por simplificar el problema, será el valor dado por el dispositivo de control en ese momento. Esto podría implicar giros bruscos que se podrían controlar mediante software. Aunque, como el aprendizaje será supervisado y el control será realista, la red de neuronas debería aprender a realizar los giros tal y como los realice el conductor.

Se puede observar un diagrama de este sistema en la Figura 3.2.


Figura 3.2: Diagrama del sistema de control y obtención de datos de entrenamiento.

*El vehículo se controla utilizando un dispositivo de control, como un volante, y visualizando vídeo en tiempo real desde un ordenador. El ordenador de control y el vehículo están conectados de forma inalámbrica, compartiendo datos en ambos sentidos. Se guardan imágenes etiquetadas con el valor de giro marcado por el dispositivo de control en ese momento.*

Aunque el sistema de obtención de datos de entrenamiento podría estar preparado para obtener la etiqueta de velocidad, este proyecto se centrará en el control autónomo de la dirección del vehículo.

## 3.2. Evaluación de métodos, medios y herramientas

En este apartado se realizará una evaluación de diferentes métodos, medios y herramientas que pueden ser utilizados para abordar el problema descrito. Para realizar la evaluación se expondrán características de las opciones expuestas y las razones que han llevado a tomar una decisión.

### 3.2.1. Sistema de conducción autónoma

El sistema de conducción se encargará de mantener una velocidad determinada y controlar de forma autónoma la dirección del vehículo. El agente autónomo desarrollado debe recorrer un circuito delimitado por dos líneas de carretera sin salirse.

Como se ha indicado anteriormente, este sistema se divide en dos tareas. La primera de ellas es la extracción de características de las líneas de la carretera. Uno de los objetivos de este proyecto es comprobar si un *Autoencoder* es una técnica viable para realizar esta tarea.

Los *Autoencoders* permiten reducir la entrada en un vector de características, llamado vector latente. Estas características están relacionadas con la imagen de entrada de forma que permiten reconstruirla. En caso de querer reconstruir solo una parte, por ejemplo, las líneas de la carretera, el vector latente estaría formado por características útiles de ese elemento. Esta aplicación de los *Autoencoders* ha sido mencionada en el estado del arte con el nombre de segmentación o extracción de una zona de interés.

Abordar la tarea de extracción de las líneas de la carretera en un escenario real sería muy complicado dado que es necesario diferenciar este elemento del resto en la escena. Para que sea posible realizar esta tarea sería necesario entrenar un *Autoencoder* en un escenario artificial que permita diferenciar los elementos. De esta forma se podrán generar imágenes aumentadas basadas en escenas reales como entrada del entrenamiento, e imágenes con únicamente las líneas de la carretera como salida esperada. Tras el entrenamiento, podría utilizarse el *Autoencoder* en el escenario real para aislar las líneas.

Una de las grandes ventajas de esta red es la reducción de la dimensionalidad. Utilizando un *Autoencoder* se obtendría un vector de características que puede ser de un tamaño mucho menor a la imagen de entrada. Esto provocaría que la red que aprende a realizar la conducción sea de menor tamaño.

En este trabajo se estudiará el uso de este tipo de red para abordar la tarea de extracción de características.

Para la segunda tarea del sistema, el aprendizaje de la conducción, se utilizará una red densamente conectada. Esta red de decisión será entrenada con las características extraídas y el valor del eje de dirección del vehículo.

Teniendo este sistema, una vez se entrene el *Autoencoder* en el escenario deseado, solo sería necesario entrenar la red de decisión. Esto último permite, por ejemplo, entrenar diferentes estilos de conducción en un tiempo menor en comparación con la técnica descrita en el artículo *End to End Learning for Self-Driving Cars* [28] en la que implicaría reentrenar toda la red.

### 3.2.2. *Framework* de Redes de Neuronas Artificiales

Existen diferentes ***frameworks*** [31] que ofrecen todo lo necesario para trabajar con redes de neuronas, en este caso se proponen los siguientes: Tensorflow [32], Keras [33] y PyTorch [34], los más utilizados tanto por profesionales como por principiantes de este campo.

**Keras** no es en sí mismo un *framework*, sino que es una interfaz (API) de alto nivel cuyo objetivo principal es simplificar la complejidad de algunas librerías de *Deep Learning*, inicialmente Theano [35] aunque ya en desuso. Ya que es una herramienta más fácil de usar que el resto, existe mucha documentación y libros que utilizan esta interfaz, por ejemplo, el libro del propio creador de Keras [19]. Esta herramienta está basada en

bloques de construcción, de manera que crear modelos es una tarea más sencilla. Sin embargo, esto último puede ser limitante en algunos casos.

**Tensorflow** dispone de una interfaz tanto de alto como de bajo nivel, por lo que ofrece una flexibilidad muy alta. Ya que ofrece distintos niveles de abstracción, la construcción de modelos es sencilla, y en caso de ser necesario, permite acceder a las funciones necesarias a más bajo nivel. Por último, Tensorflow dispone de una gran variedad de funcionalidades dentro del campo de la computación numérica (no se limita al aprendizaje automático).

**PyTorch** es una librería que ofrece una interfaz de bajo nivel. Está enfocada para ser utilizada con expresiones matemáticas que modifiquen cualquier parámetro de un modelo. En los últimos años ha ganado interés entre los investigadores académicos y su uso en aplicaciones comerciales de *Deep Learning*.

La gran popularidad de Keras y Tensorflow ha provocado que finalmente la interfaz de alto nivel, Keras, haya sido incluida directamente sobre Tensorflow [36], [37]. Esto ha permitido una mayor compatibilidad entre ambas y, además, la posibilidad de utilizarlas al mismo tiempo en caso de necesitar la modificación de aspectos más complejos.

PyTorch tiene muchas similitudes con Tensorflow en cuanto a rendimiento y funcionalidades en el campo de D*eep Learning*. Sin embargo, no es tan conocido como los otros, teniendo una menor cantidad de recursos de documentación.

Por lo tanto, debido a la enorme cantidad de recursos existentes para aprender a utilizar Keras, se ha decidido utilizar esta interfaz sobre Tensorflow para el desarrollo de todas las redes de neuronas de este proyecto.

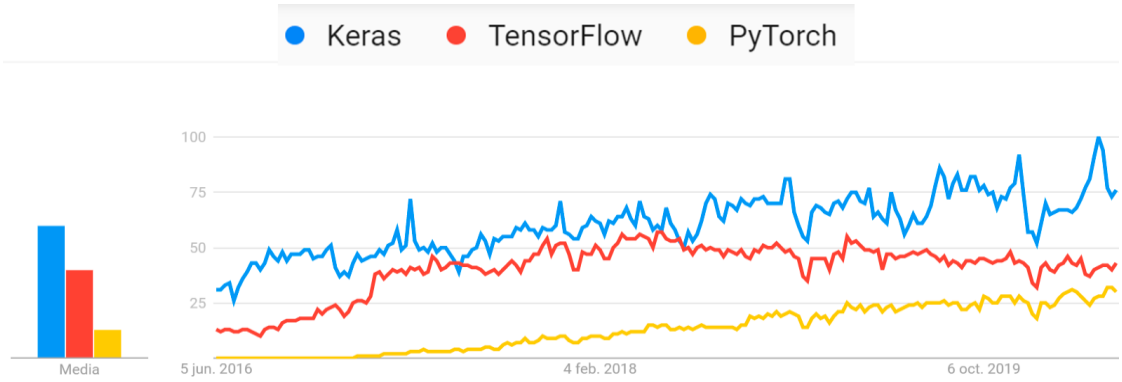



Figura 3.3: Popularidad en los últimos años de librerías de *Deep Learning*.

*Este gráfico muestra el interés a lo largo del tiempo de las tres librerías de redes de neuronas comparadas. Fuente: Google Trends.*

### 3.2.3. Visión artificial

Para abordar la tarea de extracción de una zona de interés mediante *Autoencoders* es necesario realizar procesamiento de imágenes.

La visión artificial por ordenador es un área científica que trata con métodos para capturar, procesar y analizar imágenes con el objetivo de obtener información tratable por un ordenador. Existen diferentes librerías con herramientas útiles para la visión artificial, se proponen las siguientes: OpenCV [38] (Open Source Computer Vision Library) y SimpleCV.

OpenCV es una de las librerías de visión artificial más conocidas y utilizadas. Gracias a ello, la cantidad de documentación y recursos existentes en la actualidad es muy abundante. Esta librería está programada en C/C++ por lo que ofrece una alta velocidad de procesamiento, aunque existen interfaces para otros lenguajes como Python.

SimpleCV es otra de las librerías de visión artificial existentes. Aunque es menos conocida y utilizada, ofrece acceso a las herramientas de visión artificial de una forma más sencilla, sin necesidad de tener mucho conocimiento sobre procesamiento de imágenes. La librería está programada sobre Python, un lenguaje cuya curva de aprendizaje es mucho menor que la de otros lenguajes como C.

Aunque SimpleCV ofrezca la posibilidad de aplicar visión artificial más fácilmente, se toma la decisión de utilizar OpenCV por la gran cantidad de recursos existentes para dicha librería y por la mayor flexibilidad que ofrece. Además, OpenCV proporciona una mayor eficiencia, una característica clave para el desarrollo sobre un SBC.

### 3.2.4. Vehículo a escala reducida

En este apartado se evaluarán los componentes principales del vehículo y se tomará una decisión para formar el vehículo del proyecto.

#### 3.2.4.1. Procesamiento - *Single Board Computer* (SBC)

En este apartado se evaluarán diferentes placas para realizar el procesamiento necesario. En la Tabla 3.1 se muestran las características de los componentes propuestos.

Tabla 3.1: Características principales de los SBC propuestos.

| | **Características principales** | | | | |
|---|---|---|---|---|---|
| **SBC** | **CPU** | **GPU** | **Alimentación recomendada** | **Peso** | **PVP** |
| RPi 3 B | 1,2 GHz – 4 núcleos | VideoCore IV 400 MHz | 5 V – 2,5 A | 50 g | 35 € |
| RPi 3 B + | 1,4 GHz – 4 núcleos | VideoCore IV 400 MHz | 5 V – 2,5 A | 50 g | 45 € |
| NVIDIA Jetson Nano | 1,43 GHz – 4 núcleos | Maxwell - 128 núcleos | 5 V – 4 A | 250 g | 169 € |
| Google Coral Development Board* | NXP i.MX 8M SOC* | GC7000 Lite Graphics* | 5 V – 3 A | 136 g | 159 € |

***Nota.** La Google Coral Board incluye un coprocesador TPU para acelerar tareas de inteligencia artificial.*

También se ha tenido en cuenta el uso de un microcontrolador como el Arduino Uno R3. Esta placa permite ejecutar pequeñas aplicaciones en C++ y está preparada para funcionar directamente con componentes electrónicos (como motores, servomotores...) ya que tiene la capacidad de tratar con señales PWM. Sin embargo, no sería capaz de realizar tareas avanzadas como las que se quieren abordar y sería necesario combinarla con alguno de los SBC propuestos. Se tratará de nuevo con esta placa más adelante.

A continuación se presentan las ventajas y desventajas de las opciones propuestas:

- **Raspberry Pi 3:** Cuenta con un procesador de cuatro núcleos con una velocidad de 1,2 o 1,4 GHz dependiendo del modelo. Aunque la potencia que ofrece no es

muy elevada, es una de las SBC más conocidas y utilizadas. Existe mucha documentación y recursos compatibles para este dispositivo, una gran ventaja para enfrentarse a todas las necesidades que surgirán en el proyecto. Otra de sus ventajas es la posibilidad de conectar una cámara de la propia marca a través de un puerto dedicado, consiguiendo capturar imágenes de forma más eficiente.

- **NVIDIA Jetson Nano:** Es una alternativa muy popular a Raspberry Pi, una de sus principales ventajas es la potencia de la que dispone. Tiene un procesador de cuatro núcleos a 1,43 GHz, además de una GPU Maxwell de 128 núcleos. Esta placa está orientada a la ejecución de aplicaciones de Inteligencia Artificial, algo que será necesario en este proyecto. Su precio es elevado, pero supone una alternativa mucho más potente. Sin embargo, también podría incluir algunas dificultades, como problemas de alimentación o requerir una mayor curva de aprendizaje.
- **Google Coral Development Board**: Esta placa dispone de un coprocesador TPU que disminuye el tiempo de inferencia en tareas de *Deep Learning*. Además, incluye un sistema operativo basado en Debian, por lo que tiene muy buena compatibilidad con muchas de las herramientas necesarias, por ejemplo, con Tensorflow Lite. Al igual que la placa de NVIDIA tiene un precio elevado, como contrapartida a las ventajas que puede aportar.

(a) Raspberry Pi 3 B+ (b) NVIDIA Jetson Nano (c) Google Coral Development Board

Figura 3.4: Placas de procesamiento (SBC) propuestas.

Aprovechando la posición que tiene Raspberry en el mercado tanto para obtener reemplazos fácilmente como por toda la documentación y recursos existentes, finalmente se decide utilizar la Raspberry Pi 3B+, de 1,4 GHz, aunque también se utilizará una Raspberry Pi 3B para realizar pruebas.

Además, en las primeras fases no se quiere complicar en exceso el trabajo para reducir riesgos. Utilizar la NVIDIA Jetson Nano o la Google Coral Board implicaría la necesidad de utilizar componentes más avanzados de alimentación, baterías más pesadas o problemas de compatibilidad por su menor popularidad en la comunidad. Además, el peso de la placa de NVIDIA haría necesario un vehículo bastante sólido y con mejores componentes para el desplazamiento. Se podría utilizar uno de estos SBC en fases más avanzadas, se propondrá esta opción en el apartado de trabajos futuros.

#### 3.2.4.2. Vehículo a escala

Para realizar el proyecto se ha decidido escoger un coche a escala reducida. Actualmente existen muchos kits que ofrecen las características necesarias. Inicialmente se proponen dos kits, ELEGOO UNO R3 Kit V3 y SUNFOUNDER Raspberry Pi Smart Video Car Kit.

(a) ELEGOO UNO R3 Kit V3

(b) SUNFOUNDER Raspberry Pi Smart Video Car Kit

Figura 3.5: Kits propuestos para el vehículo a escala.

En el caso de la marca ELEGOO (Imagen (a) en la Figura 3.5), muy conocida en el mercado relacionado con Arduino, ofrece un kit con el que se puede construir un coche muy completo. Las principales características de este kit son las siguientes:

- Este coche tiene cuatro ruedas motorizadas fijas. Eso implica que los giros se realizan cambiando la velocidad en los motores. Los motores incluidos tienen una reductora 1:48, se tratará más adelante con este componente.
- La dimensiones del vehículo son: 23 cm de largo y 13 cm de ancho. Se podría decir que la escala del vehículo con respecto al tamaño de uno medio es 1:16.
- Incluye un microcontrolador equivalente a la Arduino Uno R3, del que se ha hablado en el apartado anterior.
- El kit está compuesto por otros componentes que podrían ser interesantes para realizar alguna ampliación a la tarea, u otras tareas, como un sensor de ultrasonido o sensores de seguimiento de línea.

La marca SUNFOUNDER (Imagen (b) en la Figura 3.5) ofrece un kit que permite construir un coche algo más avanzado. Las principales características son las siguientes:

- La tracción del vehículo se realiza mediante dos motores con reductora 1:48 en las ruedas traseras. La dirección se controla a través de las ruedas delanteras mediante un servomotor que mueve un eje y dirige las ruedas hacía donde se desea girar.
- Las dimensiones del vehículo son: 23 cm de largo y 15 cm de ancho. Al igual que el modelo de Elegoo, se podría decir que la escala es de 1:16.
- Incluye una placa que permite controlar señales PWM. Esta se conecta a una placa de procesamiento a través de una comunicación I2C[1].
- Este kit incluye una cámara USB.

Ambos kits podrían ser válidos para desarrollar el proyecto, pero sería necesario adquirir una placa adicional que se encargue del procesamiento, el SBC seleccionado en el apartado anterior.

[1] La comunicación I2C se realiza a través de dos líneas de comunicación. La primera de ellas es la línea de datos SDA. La segunda se encarga de llevar la señal de reloj que sincroniza los datos de la primera línea.

En el caso del kit de ELEGOO sería necesario conectar el microcontrolador incluido a la Raspberry para realizar el procesamiento más complejo, ya que por sí misma solo es capaz de ejecutar pequeños programas.

El kit de SUNFOUNDER no incluye una placa capaz de realizar ningún procesamiento, únicamente incluye un controlador de señales PWM (PCA9685) para conectar con una Raspberry Pi. Una gran ventaja de este kit es que está preparado para el uso con este SBC.

Finalmente se ha seleccionado el kit de SUNFOUNDER. Este kit ya está preparado para funcionar con Raspberry, tanto a nivel de conexión como en cuanto al espacio disponible para incorporarlo. Además, el giro en este vehículo se realiza mediante un servomotor que mueve un eje de dirección en las ruedas delanteras, asemejándose al control que realizan los coches reales y, por tanto, resulta más interesante para el proyecto.

#### 3.2.4.3. Cámara

Una de las funcionalidades básicas necesarias es el uso de una cámara montada en la parte frontal del vehículo que permita obtener imágenes.

En la Tabla 3.2 se muestran las opciones propuestas y las características principales necesarias para tomar una decisión.

Tabla 3.2: Características principales de las cámaras propuestas.

| | **Características principales** | | |
|---|---|---|---|
| **Cámara** | **Conexión** | **Megapíxeles** | **Ángulo de visión** |
| Webcam Convencional | USB 2.0 | 5 MP | - |
| Pi Camera Rev 1.3 | Directa a GPU | 5 MP | 62,2º |
| Pi Camera Rev 2.2 | Directa a GPU | 5 MP | 160º |


(a) Webcam convencional (b) Pi Camera Rev 1.3 (c) Pi Camera Rev 2.2

Figura 3.6: Modelos de cámaras propuestas.

El kit del vehículo escogido incluye la webcam convencional mencionada, por lo que no sería necesario adquirirla. Esta cámara podría ser suficiente para el desarrollo, pero su conectividad por USB podría provocar retardos a la hora de obtener imágenes.

Como ya se ha indicado anteriormente, el SBC seleccionado (Raspberry Pi 3) dispone de un puerto específico en el que se puede conectar una cámara. Este tiene la ventaja de conectar directamente con GPU (unidad gráfica de procesamiento), permitiendo que la CPU esté libre para otros procesos y que la toma de imágenes sea más eficiente.

Siendo la eficiencia un aspecto muy importante al desarrollar sobre esta placa, se decide utilizar una Pi Camera para aprovechar la gran ventaja ya mencionada. Existen

diferentes versiones de esta cámara, se ha diferenciado entre la versión con una lente estándar (Rev 1.3) y una lente gran angular (Rev 2.2). Ambos módulos han sido utilizados durante el proyecto, como se especificará en cada fase.

#### 3.2.4.4. Alimentación

La alimentación del sistema ha sido otro de los puntos importantes del proyecto. Se proponen dos tipos diferentes de alimentación por batería.

Tabla 3.3: Características principales de las baterías propuestas.

| | Características principales (por unidad) | | |
|---|---|---|---|
| **Batería** | **Tensión nominal** | **Capacidad** | **Peso** |
| Ion Litio – NCR18650B | 3,6 V | 3.400 mAh | 48,5 g |
| Níquel–Metal hidruro - AA | 1,2 V | 1.600 mAh | 27,2 g |

- Ion Litio – NCR18650B: Estas son las baterías propuestas por el kit escogido y, por tanto, las que han sido utilizadas inicialmente. Este tipo de baterías disponen de una gran densidad de energía respecto a su tamaño y peso, y una tensión nominal de 3,6 V por unidad. Además, tienen una vida útil superior a otro tipo de baterías. Uno de los problemas que tienen es que son más inestables que el resto en caso de cortocircuito o carga incorrecta, por lo que suponen un riesgo. Se utilizarían dos baterías de este tipo, ofreciendo un voltaje total de 7,2 V.
- Níquel – Metal hidruro - AA: Este tipo de baterías ofrece una densidad de energía inferior en comparación con las anteriores, por lo que es necesario un mayor volumen para ofrecer la misma cantidad de energía. La tensión nominal por unidad es de 1,2 V. Estas tienen una eficiencia inferior, degradándose más rápido con el tiempo y disminuyendo la tensión durante la descarga. La principal ventaja de estas baterías es que son más seguras que las de Ion Litio. Se utilizarían entre 6 y 8 baterías de este tipo, ofreciendo entre 7,2 V y 9,6 V.

(a) Ion Litio – NCR18650B


(b) Níquel – Metal hidruro AA

Figura 3.7: Tipos de baterías propuestas.

Para las pruebas iniciales se han utilizado las baterías de Ion Litio. Debido al elevado consumo del vehículo y de sus componentes, han surgido problemas de falta de corriente. Añadiendo esto a los otros problemas expuestos, se decidió utilizar las de Níquel – Metal hidruro AA, principalmente por la seguridad que ofrecen. En concreto, han sido necesarias 8 unidades de estas baterías para desarrollar correctamente el proyecto dado que con 6 unidades los problemas de alimentación persistían.

Uno de los componentes del vehículo se encarga de convertir la corriente de entrada procedente de las baterías a unos niveles que puedan ser utilizados por el sistema. Este componente es el *Step Down DC-DC Converter Module*. El módulo incluido en el kit seleccionado está basado en el chip XL1509.

La Raspberry Pi 3 necesita una alimentación de 5 V y entre 1,5 y 2,5 A para funcionar correctamente, por lo que este componente es bastante importante. Debido a los problemas de alimentación experimentados en un comienzo, se estudia una alternativa (LM2596) por si pudiera aportar alguna ventaja. En la Tabla 3.4 se muestra una comparativa de características.

Tabla 3.4: Características principales de los componentes de alimentación propuestos.

| | **Características principales** | |
|---|---|---|
| **Componente** | **Tensión mínima** | **Alimentación de salida** |
| XL1509 | 6,5 V | 5 V – 2 A |
| LM2596 | 6 V | 5 V – 3 A |

(a) Componente basado en el XL1509 (b) Componente LM2596

Figura 3.8: Componentes de alimentación propuestos.

- El módulo basado en el chip XL1509 necesita una entrada de 6,5 V para generar en la salida al menos 5 V y una corriente constante de 2 A. La salida debe ser muy estable dado que la corriente que ofrece se encuentra, aunque dentro en el rango de funcionamiento, por debajo de la máxima recomendada para alimentar al SBC.
- El módulo LM2596 necesita una entrada de al menos 6 V para funcionar correctamente. En la salida ofrece una alimentación constante de 5 V y 3 A.

El primer módulo mencionado está incluido en el kit del vehículo seleccionado, por lo que no sería necesario adquirirlo aparte y se presupone que debería funcionar correctamente. Uno de los problemas que se tendría con el módulo LM2596 es que es físicamente diferente, por lo que sería necesario encontrar la forma de fijarlo en el chasis.

En principio se utiliza el módulo incluido (XL1509) ya que al escoger como alimentación las 8 baterías de NiMh no se vuelven a reproducir los problemas de alimentación. El segundo módulo queda disponible como material experimental para futuras necesidades.

#### 3.2.4.5. Motores de tracción

El movimiento del vehículo se realiza mediante dos motores, uno para cada rueda trasera, asemejándose ligeramente a la realidad. El kit seleccionado dispone de estos motores, pero se proponen otras opciones.

Será necesario que aporten la velocidad en un rango de interés. Deben ser lo suficientemente lentos como para que el vehículo pueda realizar inferencia en redes de

neuronas sin quedarse parado por falta de torque[2]. Aunque también interesa que alcancen velocidades altas.

Para tomar una decisión concluyente sería necesario conocer el tiempo de inferencia con el que calcular un rango de velocidad a la que se debería mover el vehículo. No se toma una decisión en este apartado, ya que dependerá de los resultados obtenidos en el apartado de diseño y experimentación final. Hasta entonces, serán utilizados los motores que incluye el kit seleccionado, los TT Motor DC 1:48.

#### 3.2.4.6. Comunicación inalámbrica

Para enviar comandos de ejecución a la Raspberry ha sido necesario utilizar algún método de comunicación entre un ordenador y el SBC. Inicialmente todo se realizaba a través de conexión directa con HDMI entre un monitor externo y la Raspberry. Ha sido necesario utilizar comunicación inalámbrica para permitir su libre movimiento. Se proponen dos métodos:

Tabla 3.5: Características principales de los métodos de comunicación propuestos.

| | **Características principales** | | |
|---|---|---|---|
| **Método** | **Frecuencia** | **Velocidad máxima de transmisión** | **Alcance teórico en abierto** |
| NRF24L01 | 2,4 – 2,5 GHz | 250 kbps – 2 Mbps | 500 metros |
| Adaptador wifi USB | 2,4 / 5 GHz | 150 / 433 Mbps | 90 / 30 metros |


(a) Transceptor NRF24L01 (b) Adaptador wifi USB

Figura 3.9: Métodos de comunicación propuestos.

- Uso del chip NRF24L01. Es un transceptor, es decir, cuenta con transmisor y receptor. Funciona en una frecuencia entre 2,4 GHz y 2,5 GHz y es posible seleccionar algunos canales libres de interferencias de redes wifi. Puede alcanzar velocidades de transmisión de entre 250 Kbps y 2 Mbps. Tiene un alcance teórico de hasta 500 metros, aunque en la práctica este podría ser mucho menor.
- Uso de un adaptador wifi USB. Es la conexión utilizada a diario por dispositivos inalámbricos. El adaptador trabajando en la frecuencia de radio 2,4 GHz puede alcanzar velocidades de transmisión de hasta 150 Mbps. Aunque con menor alcance, también se podría utilizar la frecuencia de 5 GHz para obtener mayor velocidad de transmisión.

[2] El torque es la capacidad de desarrollar fuerza sobre un eje. A mayor torque, el vehículo tiene la capacidad de moverse a velocidades más bajas.

Aunque el chip NRF24L01 ofrece algunas ventajas como su alcance, bajo coste o trabajar en una frecuencia libre de interferencias, su velocidad de transmisión sería válida para comunicación de comandos, pero no para la transmisión de vídeo en tiempo real.

Se decide utilizar la conexión por wifi ya que ofrece la velocidad de transmisión necesaria. Además, este será mucho más sencillo de utilizar ya que tan solo será necesario crear una red wifi privada para los dispositivos involucrados. Se instala en la Raspberry el adaptador wifi USB descrito, mejorando el alcance del dispositivo que tiene por defecto el SBC.

#### 3.2.4.7. Control del vehículo

Para controlar el vehículo es necesario algún dispositivo que facilite la conducción y que permita realizar una conducción realista. Para abordar esta tarea se proponen diferentes dispositivos:

Tabla 3.6: Características principales de los dispositivos de control propuestos.

| | **Características principales** | | | |
|---|---|---|---|---|
| **Dispositivo** | **Conexión** | **Angulo de rotación** | **Resistencia de giro** | **Precio** |
| Mando Xbox One | Bluetooth | - | - | 47 € |
| Thrustmaster T80 | USB | 240º | Lineal | 98 € |
| Logitech G29 | USB | 900º | No lineal – *Force feedback* | 300 € |

(a) Mando de Xbox One (b) Thrustmaster T80 (c) Logitech G29

Figura 3.10: Dispositivos de control propuestos.

- Mando de la consola Xbox One: Este control es inalámbrico, permitiendo conectarse a un ordenador a través de bluetooth. Las características que interesan son los dos gatillos y dos ejes que permiten la aceleración y giro con total precisión.
- Kit de conducción Thrustmaster T80: Este kit de conducción dispone de volante y pedales. El volante dispone de un ángulo de rotación de 240º y dispone de dos pedales. Dispone de un mecanismo de auto centrado y una resistencia de giro lineal. Este kit destaca por las características que ofrece en relación con su precio.
- Kit de conducción Logitech G29: Este kit de conducción dispone de volante y pedales. El volante tiene un ángulo de rotación de 900º y dispone de 3 pedales (embrague, freno y acelerador). El volante dispone de un par de motores que ofrecen auto centrado, retroalimentación y resistencia de giro no lineal, dando realismo a la conducción. Este es uno de los kits de conducción más populares del mercado.

Algo muy importante a tener en cuenta al seleccionar el dispositivo de control es su compatibilidad con los programas que se quieren utilizar (PyGame). Es por este motivo por lo que se duda de la capacidad del volante Thrustmaster T80. Además de tener un ángulo de giro reducido, no se dispone de información clara sobre la compatibilidad de este volante con la librería de control que será utilizada.

Tanto el mando de Xbox One como el kit de conducción Logitech G29 disponen de la compatibilidad deseada. Además, existen programas que permiten configurar fácilmente este volante, como la fuerza de auto centrado o limitar el ángulo de giro.

Se decide utilizar como control realista el Logitech G29. Aunque tiene un precio elevado, con este volante se asegura la compatibilidad y la funcionalidad buscada. En las primeras fases del proyecto se utilizará el control mediante el mando de la Xbox One.

### 3.3. Especificación de requisitos

En este apartado se realiza la especificación de requisitos del proyecto. Estos han sido elaborados teniendo en cuenta la descripción del problema y la evaluación de métodos, medios y herramientas anterior.

En el capítulo de validación del proyecto se realizarán las pruebas necesarias para validar el cumplimiento de cada requisito.

Cada requisito quedará definido en una tabla con los siguientes campos:

- **Identificador**: Nombre que identifica unívocamente al requisito mediante una nomenclatura específica.
    - **RF-XX** (requisito funcional número XX)
    - **RNF-XX** (requisito no funcional número XX)
- **Nombre**: Nombre descriptivo del requisito.
- **Grupo**: Se han agrupado los requisitos en función del sistema al que hacen referencia. Se han establecido los siguientes grupos de requisitos:
    - **Entorno de trabajo (E)**: Hace referencia a aspectos del escenario artificial, escenario objetivo y circuito utilizado para entrenar las redes de neuronas.
    - **Vehículo / Sistema de control (V/SC):** Hace referencia al vehículo y al sistema de control.
    - **Sistema de conducción autónoma (SCA):** Hace referencia a todo lo relacionado con redes de neuronas y el agente desarrollado.
- **Necesidad**: Indica la importancia del requisito para el proyecto. La necesidad se indica por uno de los siguientes valores: ALTA (imprescindible), MEDIA (deseable) o BAJA (opcional).
- **Prioridad**: Indica la antelación con la que el requisito se debe cumplir. Este campo sirve para tener en cuenta las necesidades inminentes durante el desarrollo del proyecto. La prioridad se indica por uno de los siguientes valores: ALTA, MEDIA o BAJA.
- **Estabilidad**: Indica la probabilidad de que este requisito no cambie durante el desarrollo del proyecto. Puede tomar uno de los siguientes valores: ALTA

(raramente cambiaría), MEDIA (podría cambiar) o BAJA (probablemente cambie).

- **Descripción**: Definición de la necesidad a la que se refiere el requisito.
- **Comentarios:** Aporta algún detalle adicional del requisito. Este campo es opcional y solo aparecerá en aquellos requisitos en los que se considere necesario.

Los requisitos estarán clasificados principalmente en dos grupos:

- **Requisitos funcionales**: Son declaraciones de las funciones que provee el sistema.
- **Requisitos no funcionales:** Son aquellos que no declaran nuevas funcionalidades del sistema, sino propiedades o restricciones de este. Estas propiedades o restricciones están relacionadas con el rendimiento, disponibilidad, estabilidad o funcionalidad del sistema.

En la Tabla 3.7 se puede observar la plantilla que será utilizada para la definición de cada requisito.

Tabla 3.7: Plantilla de definición de requisitos.

| **RF-XX / RNF-XX** | | | |
|---|---|---|---|
| **Nombre** | | | |
| **Necesidad** | | **Prioridad** | |
| **Estabilidad** | | **Grupo** | |
| **Descripción** | | | |
| **Comentarios** | *[Campo opcional]* | | |

### 3.3.1. Requisitos funcionales

Tabla 3.8: RF-01

| **RF-01** | | | |
|---|---|---|---|
| **Nombre** | Conducción autónoma | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | SCA |
| **Descripción** | El agente desarrollado debe ser capaz de seguir una carretera delimitada por dos líneas. Se debe tratar de maximizar el realismo del entorno en el que se realiza la conducción y del estilo de conducción alcanzado. | | |

Tabla 3.9: RF-02

| **RF-02** | | | |
|---|---|---|---|
| **Nombre** | Control remoto | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | El vehículo debe permitir su control de forma remota mediante un sistema de control. Dicho sistema debe permitir el control de la velocidad y dirección del vehículo y la visualización necesaria para realizar dicho control. | | |

Tabla 3.10: RF-03

| RF-03 | | | |
|---|---|---|---|
| **Nombre** | Funcionalidad de velocidad constante | | |
| **Necesidad** | MEDIA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | El sistema de control debe permitir establecer una velocidad constante prefijada en el vehículo. | | |

Tabla 3.11: RF-04

| RF-04 | | | |
|---|---|---|---|
| **Nombre** | Obtención de datos de entrenamiento – etiqueta de dirección | | |
| **Necesidad** | ALTA | **Prioridad** | MEDIA |
| **Estabilidad** | ALTA | **Grupo** | V/SC |
| **Descripción** | El sistema de control debe ofrecer la opción de guardar secuencias de conducción junto a una etiqueta con el valor de dirección. | | |

Tabla 3.12: RF-05

| RF-05 | | | |
|---|---|---|---|
| **Nombre** | HUD del vídeo en tiempo real | | |
| **Necesidad** | BAJA | **Prioridad** | BAJA |
| **Estabilidad** | NAJA | **Grupo** | V/SC |
| **Descripción** | El vídeo en tiempo real del sistema de control debe ofrecer información sobre la velocidad y giro del vehículo. | | |

### 3.3.2. Requisitos no funcionales

Tabla 3.13: RNF-01

| RNF-01 | | | |
|---|---|---|---|
| **Nombre** | Entrada al sistema de conducción autónoma | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | ALTA | **Grupo** | SCA |
| **Descripción** | La única entrada del sistema de conducción debe ser una imagen tomada desde el vehículo. | | |

Tabla 3.14: RNF-02

| RNF-02 | | | |
|---|---|---|---|
| **Nombre** | Extracción de características de las líneas de la carretera | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | SCA |
| **Descripción** | La tarea de extracción de características de las líneas de la carretera la debe implementar una red de neuronas, en concreto, un *Autoencoder*. Esta red debe ofrecer un vector de características correspondiente a dichas líneas. | | |

Tabla 3.15: RNF-03

| **RNF-03** | | | |
|---|---|---|---|
| **Nombre** | Control autónomo | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | SCA |
| **Descripción** | El control autónomo del vehículo lo debe implementar una red de neuronas. La entrada de esta red será el vector de características de las líneas de carretera. Esta red debe controlar la dirección del vehículo. | | |

Tabla 3.16: RNF-04

| **RNF-04** | | | |
|---|---|---|---|
| **Nombre** | Generalización y robustez | | |
| **Necesidad** | MEDIA | **Prioridad** | BAJA |
| **Estabilidad** | MEDIA | **Grupo** | SCA |
| **Descripción** | El sistema de conducción autónoma debe generalizar y ser robusto en alguna medida. | | |
| **Comentarios** | *Este requisito es un tanto impreciso. Con el sistema completo se deben realizar pruebas que estudien el alcance del sistema bajo estos términos.* | | |

Tabla 3.17: RNF-05

| **RNF-05** | | | |
|---|---|---|---|
| **Nombre** | Procesamiento centralizado | | |
| **Necesidad** | ALTA | **Prioridad** | MEDIA |
| **Estabilidad** | MEDIA | **Grupo** | SCA |
| **Descripción** | El procesamiento necesario para realizar la tarea de conducción autónoma debe estar centralizado en el SBC utilizado. | | |

Tabla 3.18: RNF-06

| **RNF-06** | | | |
|---|---|---|---|
| **Nombre** | Cámara del vehículo | | |
| **Necesidad** | ALTA | **Prioridad** | MEDIA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | El vehículo debe disponer de una cámara en su parte frontal. La orientación y características de esta debe permitir visualizar las líneas que limitan la carretera. | | |

Tabla 3.19: RNF-07

| **RNF-07** | | | |
|---|---|---|---|
| **Nombre** | Control manual realista | | |
| **Necesidad** | MEDIA | **Prioridad** | MEDIA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | El control manual del vehículo debe ser realista. | | |
| **Comentarios** | *Para precisar, el realismo en este caso se refiere a que la forma de conducir el vehículo en entrenamiento sea parecida a la realidad. Se mide este concepto en cuanto al dispositivo de control y la forma de visualizar la carretera utilizada por el operario.* | | |

Tabla 3.20: RNF-08

| **RNF-08** | | | |
|---|---|---|---|
| **Nombre** | Capacidad de giro | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | El vehículo debe ser capaz de tomar curvas de hasta 90º mientras esta tenga un radio de, al menos, la longitud del vehículo. | | |
| **Comentarios** | *Este requisito es necesario para poder desarrollar circuitos de un tamaño razonable.* | | |

Tabla 3.21: RNF-09

| **RNF-09** | | | |
|---|---|---|---|
| **Nombre** | Rendimiento del vídeo en tiempo real | | |
| **Necesidad** | MEDIA | **Prioridad** | ALTA |
| **Estabilidad** | BAJA | **Grupo** | V/SC |
| **Descripción** | El vídeo en tiempo real debe ser estable, ofrecer un mínimo de 10 FPS, y tener una latencia máxima de 0,1 segundos. | | |

Tabla 3.22: RNF-10

| **RNF-10** | | | |
|---|---|---|---|
| **Nombre** | Tiempo de operación del vehículo | | |
| **Necesidad** | MEDIA | **Prioridad** | MEDIA |
| **Estabilidad** | MEDIA | **Grupo** | V/SC |
| **Descripción** | Las baterías del vehículo deben permitir que este se mantenga en funcionamiento durante un mínimo de 30 minutos. Durante este periodo el vehículo debe responder de forma consistente. | | |
| **Comentarios** | *Este periodo recibe el nombre de "Tiempo de operación".* | | |

Tabla 3.23: RNF-11

| **RNF-11** | | | |
|---|---|---|---|
| **Nombre** | Espacio de operación del sistema de control | | |
| **Necesidad** | MEDIA | **Prioridad** | MEDIA |
| **Estabilidad** | BAJA | **Grupo** | V/SC |
| **Descripción** | La conexión entre el vehículo y el sistema de control debe ser estable en un rango de 5 metros. | | |
| **Comentarios** | *Este rango recibe el nombre de "Espacio de operación".* | | |

Tabla 3.24: RNF-12

| **RNF-12** | | | |
|---|---|---|---|
| **Nombre** | Materiales del escenario artificial / *chroma* | | |
| **Necesidad** | ALTA | **Prioridad** | ALTA |
| **Estabilidad** | ALTA | **Grupo** | E |
| **Descripción** | El escenario utilizado para realizar el entrenamiento del *Autoencoder* debe estar compuesto por materiales que faciliten su eliminación mediante procesamiento de imagen. | | |
| **Comentarios** | *El material tiene diferentes características que pueden complicar la tarea, como su color, brillo, rugosidad… etc.* | | |

Tabla 3.25: RNF-13

| **RNF-13** | | | |
|---|---|---|---|
| **Nombre** | Circuito | | |
| **Necesidad** | BAJA | **Prioridad** | MEDIA |
| **Estabilidad** | BAJA | **Grupo** | E |
| **Descripción** | El circuito desarrollado debe estar compuesto por piezas, de forma que sea posible formar otros recorridos. | | |

# 4. DISEÑO Y EXPERIMENTACIÓN PREVIA

En este capítulo se describe el diseño y experimentación que se ha realizado durante la mayor parte del proyecto. El contenido de este capítulo corresponde con todo el trabajo previo realizado. El diseño y la experimentación final serán desarrollados en el siguiente capítulo.

Además del contenido incluido en los capítulos de diseño, se adjuntan anexos que aportan mucha información sobre el trabajo que se ha realizado. También se ha creado un repositorio dedicado al proyecto que será descrito a continuación.

## 4.1. Repositorio del proyecto

A lo largo de la memoria se realizarán referencias a recursos que se encuentran en un repositorio dedicado a este proyecto. Este repositorio tiene el nombre de **TFGAutonomousCar** [39] y está alojado en GitHub. La dirección a dicho repositorio es **https://github.com/javiercorrochano/TFGAutonomousCar**. En él se puede encontrar lo siguiente:

- **Vídeos**: Se han elaborado vídeos durante el desarrollo del proyecto a modo de documentación del trabajo realizado. Además, este contenido facilita la comprensión de las tareas desarrolladas.

  Los vídeos se encuentran indizados por fases. Por cada fase existe un índice con los enlaces necesarios para visualizar los vídeos en la plataforma YouTube.

  Para hacer referencia a los vídeos en la memoria se ha decidido utilizar la siguiente nomenclatura: **Vídeo X.Y**, siendo X el número de la fase a la que pertenece e Y el número de vídeo dentro de dicha fase.

  En el capítulo 11, VIDEOGRAFÍA se incluye el nombre y la descripción de cada uno de los vídeos a los que se hace referencia en la memoria.

- **Pruebas de validación**: Se han incluido imágenes y vídeos sobre los resultados obtenidos en algunas de las pruebas realizadas en el apartado de validación de este proyecto.

## 4.2. Metodología

En esta sección se describe la estructura y metodología seguida durante el diseño y la experimentación del proyecto. Todas las tareas desarrolladas han sido agrupadas o clasificadas en distintas fases. El objetivo de las tareas desarrolladas en fases previas a la final ha sido el aprendizaje y la comprensión incremental del conocimiento necesario para alcanzar los objetivos marcados en el TFG.

No todas las fases son dependientes o necesariamente secuenciales. Algunas fases se han desarrollado simultáneamente en el tiempo y otras son dependientes de la finalización de una fase previa.

- ***Fase 0: Trabajo previo***: Esta fase no será documentada ya que corresponde con el trabajo inicial para orientar los objetivos del TFG. Sin embargo, esta sí aparece en la planificación del proyecto.
- ***Fase I: Seguimiento de objetos***: Esta fase corresponde con el trabajo realizado como aproximación a la visión artificial y al estudio inicial del vehículo utilizado.

- ***Fase II: Experimentación con Deep Learning***: Corresponde con el trabajo previo sobre D*eep Learning*. Esta fase ha supuesto gran parte del proyecto ya que ha sido necesario estudiar muchos aspectos de este campo.
- ***Fase III: Sistema de control y obtención de datos de entrenamiento***: Corresponde con el trabajo realizado para desarrollar un sistema que permite el control completo del vehículo y la obtención de datos de entrenamiento, necesario para desarrollar las tareas principales del proyecto.
- ***Fase IV: Tarea de extracción de las líneas de la carretera (primer entorno)***: Corresponde con el trabajo realizado para abordar la tarea de extracción de características de las líneas de la carretera en el primer entorno en el mundo real. Se realizará una aproximación a esta tarea mediante un simulador en la fase de experimentación con *Deep Learning.*
- ***Fase V: Diseño y experimentación final***: Corresponde con el trabajo realizado para abordar las tareas principales del proyecto en el entorno final.

### 4.2.1. Nomenclatura de experimentación

Durante todo el proyecto se ha seguido una metodología de experimentación concreta. En los casos en los que se ha realizado experimentación con modelos de redes de neuronas se ha establecido una nomenclatura para todos los ficheros generados. La nomenclatura se ha definido en la Tabla 4.1.

Tabla 4.1: Estructura de la nomenclatura en experimentos.

| Identificador Global | Dominio | Topología de la red | Identificador dentro del dominio | Terminación |
|---|---|---|---|---|

Los identificadores se muestran como números de cuatro dígitos, completados con ceros por la izquierda. En algunos casos se hará referencia al experimento utilizando únicamente el identificador global. *Dominio* identifica el dominio en el que se está realizando el experimento, por ejemplo, MNIST. La topología indica la estructura del modelo de red de neuronas diseñado en el experimento, definida por las abreviaturas indicadas en la Tabla 4.2.

Tabla 4.2: Abreviaturas utilizadas para representar la topología de redes de neuronas

| Descripción | Abreviatura |
|---|---|
| Capa convolucional | C |
| Capa de Max-Pooling | M |
| Capa de neuronas densamente conectadas | D |
| Capa Flatten | F |
| Capa de *Dropout* | Dr |
| Capa convolucional transpuesta | Ct |
| Capa de R*eshape* | R |

Un ejemplo de nomenclatura podría ser el siguiente: 0016-HSRL-CMCMCMFDD-0001.py. Este experimento es el número 16 globalmente, del dominio de Hand Sign Recognition Letter, es el primer experimento para dicho dominio y según la terminación,

es el fichero de código Python del experimento (se indican las terminaciones posibles en la Tabla 4.3).

Tabla 4.3: Posibles terminaciones en la nomenclatura de experimentos

| Descripción | Terminación |
|---|---|
| Fichero de código Python del experimento. | .py |
| Fichero que recoge la salida por pantalla del entrenamiento del experimento. | .txt |
| Topología y pesos guardados del experimento. | .h5 |
| Gráfica de acierto en validación y entrenamiento. | -acc.png |
| Matriz de confusión del experimento. | -cm.png |
| Gráfica de error en validación y entrenamiento del experimento. | -loss.png |

### 4.2.2. Apartados desarrollados en cada fase

En cada fase del proyecto se desarrollarán una serie de apartados. Algunos de ellos se repiten entre distintas fases ya que tratan la misma temática. Estos apartados son los siguientes:

- **Vehículo utilizado:** En este apartado se describirán todos los aspectos hardware relacionados con el vehículo utilizado, como el procesador utilizado, cámara, motores, servomotor de dirección, alimentación... etc.
- **Herramientas:** En este apartado se expondrán las diferentes herramientas software (como librerías, protocolos de comunicación, lenguajes de programación… etc.).
- **Entorno de trabajo:** En el caso de las dos últimas fases se utiliza este apartado para describir los elementos del entorno físico en el que se abordan las tareas.
- **Implementación:** En este apartado se describirá la implementación o implementaciones necesarias para el desarrollo de las tareas de la fase. Ya que en algunos casos puede ser muy técnico o extenso, se realizará un resumen o se pondrá una referencia a un anexo.
- **Experimentación:** En este apartado se describirá la experimentación realizada. Este apartado está orientado principalmente a la descripción de los experimentos realizados con redes de neuronas.

  Esta experimentación se desarrolla de la siguiente forma: Partiendo de un experimento base, se realizarán pequeños cambios en él (un cambio entre experimentos) con el objetivo de encontrar mejoras mediante las que se puedan obtener buenos modelos para la tarea que se esté realizando.
- **Conclusiones de la fase:** En este apartado se describirán las conclusiones generales obtenidas. Estas estarán relacionadas tanto con la experimentación realizada como con el conocimiento obtenido, detalles útiles para próximas fases o detalles sobre la viabilidad del proyecto.

Estos apartados no tienen por qué aparecer en todas las fases. En algunos casos no se desarrolla ninguno de ellos ya que se ha documentado el conjunto de tareas realizado,

como en la fase de experimentación con *Deep Learning* (Fase II) o el desarrollo del sistema de control (Fase III).

## 4.3. Fase I – Seguimiento de Objetos

En esta fase el objetivo es experimentar con el seguimiento de objetos mediante una herramienta de visión artificial. En primera instancia, el objetivo es lograr un programa que siga objetos (*object tracking*), en concreto, que siga una pelota verde marcando el lugar de la imagen en el que se encuentra.

En segundo lugar, se modificará ese programa para conseguir que un vehículo a escala reducida siga a ese objeto. Este seguimiento se realizará manteniendo una velocidad constante y realizando el giro necesario para alcanzar dicho objeto.

### 4.3.1. Vehículo utilizado

El vehículo utilizado es el del Kit de Sunfounder con alguna modificación. Los componentes principales del vehículo son los siguientes:

- Raspberry Pi 3B. Es el SBC encargado del procesamiento, ejecuta el sistema operativo Raspbian sobre el que ha sido instalado OpenCV y todo lo necesario para desarrollar esta tarea.
- Servomotor para la dirección. En la Tabla 4.4 se pueden observar las especificaciones de los servomotores propuestos. El servomotor incluido en el kit es el SG90, que tiene un torque de 1,8 kg/cm. Debido a algunas ineficiencias en la dirección, este servomotor se sustituye por el SG92R, cuyo torque es de 2,5 kg/cm, algo superior. Está es la única y principal diferencia entre ambos modelos. Con este cambio y un ajuste en el eje de dirección ha sido posible obtener un buen sistema de giro.

Tabla 4.4: Especificaciones de servomotores.

| Servomotor | Torque | Tensión mínima | Dimensiones |
|---|---|---|---|
| SG90 | 1,8 kg/cm | 4,8 V | 23×12,2x29 mm |
| SG92R | 2,5 kg/cm | 4,8 V | 23×12,2x27 mm |

- Dos motores de tracción. Los dos motores utilizados son motores simples con una reductora 1:48. Concretamente el modelo de este motor es el TT Motor DC 1:48, también conocido como TT Motor 130. Cada motor actúa sobre una rueda trasera del vehículo.
- Se utilizan 8 baterías AA de Níquel-Metal Hidruro, con una capacidad por unidad de 1.600 mAh. En total ofrece una tensión de 9,6 V, necesario para evitar problemas de alimentación por su bajada de tensión mientras se descargan.
- Cámara. Se utiliza la Pi Camera *rev 1.2* con la lente estándar de 62º de ángulo de visión. La cámara se monta sobre un soporte impreso en 3D y a su vez este se fija a un bloque de construcción. En la parte delantera del vehículo se fija otro bloque de construcción que permite colocar la cámara.

En la fase de diseño del vehículo final se describirán detalladamente todos los componentes.

En la Figura 4.1 y Figura 4.2 se puede observar la disposición de los componentes del vehículo en esta fase.

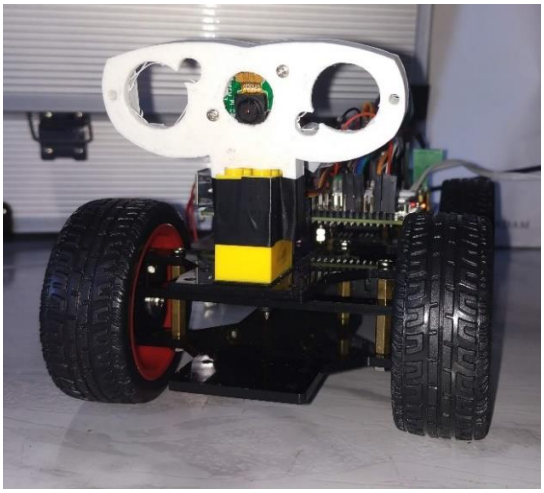


Figura 4.1: Vista frontal del vehículo.

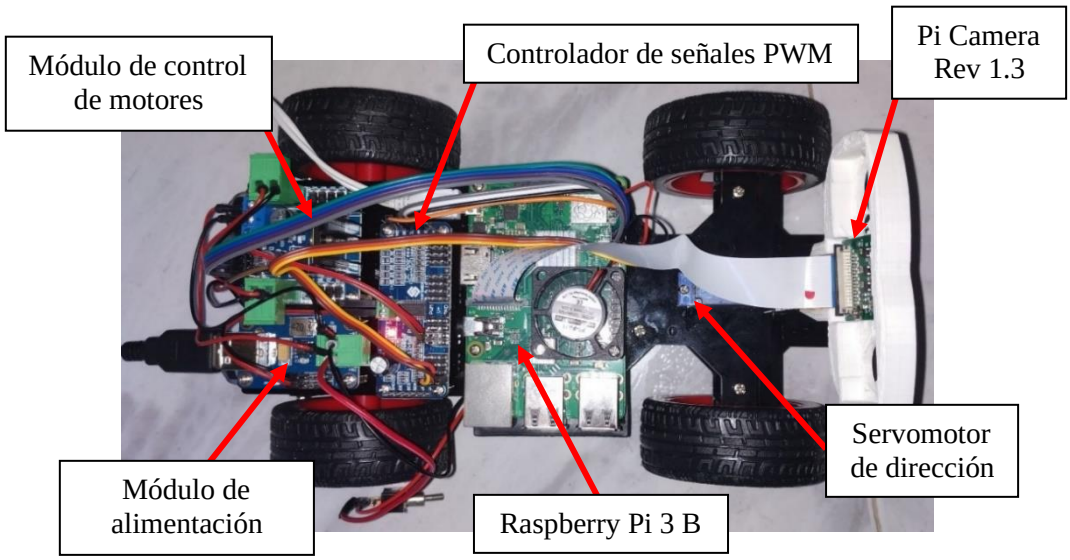



Figura 4.2: Vista superior del vehículo.

### 4.3.2. Herramientas

Para desarrollar esta tarea, ha sido necesario utilizar herramientas de visión artificial. Tal como se ha expuesto en el análisis del problema, se ha decidido utilizar OpenCV como librería de visión artificial por ordenador.

La versión que se va a utilizar de esta librería es OpenCV 4. Se incluye un anexo con aspectos relacionados con procesamiento de imagen en el que se describe el procedimiento seguido para su instalación en la Raspberry Pi (véase ANEXO II: PROCESAMIENTO DE IMAGEN, B.1. Instalación de OpenCV 4 en Raspbian-Raspberry Pi).

### 4.3.3. Implementación

Se incluye un anexo con aspectos relacionados con procesamiento de imagen en el que se describe la implementación del programa de seguimiento de objetos utilizado (véase ANEXO II: PROCESAMIENTO DE IMAGEN, B.2. Seguimiento de objetos

mediante OpenCV). En este apartado únicamente se realizará un resumen de dicha implementación, necesario para comprender esta fase del proyecto.

Esta implementación se divide en dos partes. En primer lugar, ha sido necesario un programa que detecte y marque en la imagen capturada el lugar en el que se encuentra la pelota [40]. En segundo lugar, el programa anterior se modifica para girar la dirección del coche hacia el punto en el que se encuentra el objeto.

Ya que se trata de una tarea de visión artificial, el programa utilizado hace uso de la librería OpenCV. El programa consiste en procesar cada *frame* capturado aplicando sobre él una máscara usando el espacio de color HSV. Combinando esta máscara con funciones de detección de contornos y procesamiento de imágenes se puede obtener la posición central de la pelota y su tamaño. A partir de lo anterior se puede dibujar en la imagen un círculo y una línea que marca el seguimiento.

Habiendo realizado lo anterior, es momento de abordar la tarea con el vehículo. Para ello es necesario modificar el programa para lograr el giro de la dirección. A continuación se describe el trabajo necesario para ello:

1. Una vez se tiene el punto central del objeto detectado, este punto será una coordenada (x, y) en la imagen. Solo interesa el movimiento horizontal, se puede omitir la coordenada y.
2. Teniendo la coordenada x en el rango horizontal en píxeles, por ejemplo [0, 600], sería necesario escalar este valor al rango que admite la dirección del vehículo, en este caso, [450, 550] (giro total a izquierda y giro total a derecha respectivamente, en valores de señal PWM). En ANEXO I: SEÑALES PWM se explica el funcionamiento de estas señales y su implicación en el vehículo utilizado.
3. Teniendo el valor escalado, se envía un comando al vehículo para que oriente su dirección al valor calculado. Aplicando una velocidad constante se consigue que el vehículo se desplace hacia el lugar en el que detecta la pelota.

### 4.3.4. Experimentación

La experimentación en esta fase consiste en mostrar y describir el trabajo realizado. La demostración de funcionamiento se expondrá en vídeo ya que es la mejor forma de visualizarlo. En la metodología se ha indicado dónde se encuentran los vídeos.

**Vídeo 1.1: Seguimiento de pelota en pantalla:** En este vídeo se puede observar cómo la pelota se detecta, rodeándola con un círculo, y se dibuja una línea que marca el historial de movimiento de esta.

**Vídeo 1.2: Seguimiento de pelota utilizando un vehículo:** En una primera parte de este vídeo se puede observar la detección desde la cámara y cómo la dirección del vehículo acompaña el movimiento de la pelota. En una segunda parte, se establece una velocidad constante y se le permite circular libremente. Colocando la pelota delante del vehículo se puede observar como este avanza hacia la dirección en la que es detectada.

### 4.3.5. Conclusiones

Una de las conclusiones más importantes que se han podido obtener al realizar esta tarea, es que el uso de máscaras que utilizan color puede ser muy sensible a la iluminación. Un cambio de luminosidad o de tonalidad de la fuente de luz puede hacer que la tarea deje de funcionar por completo.

Utilizar el espacio de color HSV ha facilitado en gran medida la selección de los límites de color que definen la máscara. Además, ha sido más sencillo seleccionar un rango de color algo más robusto que permitiese responder adecuadamente ante algunos cambios en el entorno. Por ejemplo, en el **Vídeo 1.2** se puede observar como la primera parte se desarrolla con iluminación cálida y la segunda con iluminación fría.

## 4.4. Fase II – Experimentación con *Deep Learning*

En esta fase el objetivo ha sido realizar experimentación con D*eep Learning* en diferentes dominios para familiarizarse con las librerías y herramientas necesarias.

Los dominios abordados han sido Dogs vs Cats, Sign Language MNIST y MNIST mediante *Autoencoders*.

En esta fase también se aborda una aproximación mediante un simulador a una de las tareas principales del proyecto, la extracción de características de las líneas de la carretera. De cara al objetivo del proyecto, esta ha sido la tarea más importante de esta fase.

En este apartado se describirán los aspectos más relevantes con los que se ha trabajado en los diferentes dominios nombrados.

### 4.4.1. Herramientas

Durante la experimentación se han utilizado diferentes herramientas y librerías. Entre ellas, algunas para el manejo y visualización de datos como Numpy, Pandas, Sklearn, Matplotlib o Seaborn. Aunque estas han tenido su importancia en el desarrollo, sin duda la herramienta principal ha sido Keras, la interfaz de redes de neuronas.

Esta herramienta ya ha sido descrita en el análisis, por lo que el único aspecto que queda por tratar es aquello relacionado con las versiones.

El ordenador utilizado durante todo el proyecto está compuesto por un procesador Intel i5 de novena generación, con 6 núcleos a 2,9 GHz, 16 GB de memoria RAM y dos tarjetas gráficas GeForce GTX 1050 Ti. El sistema operativo instalado es Ubuntu 18.04.3. Este ordenador ha sido el utilizado para entrenar todos los experimentos relacionados con *Deep Learning*.

Ya que se han utilizado componentes muy actualizados, ha sido posible utilizar versiones medianamente actualizadas. Además, para aprovechar el potencial de las tarjetas gráficas se ha utilizado la versión de CUDA 10.0, que ha permitido la instalación de Tensorflow 1.13.1. Más adelante se mostrará experimentación utilizando multi GPU. Por último, la versión de Keras utilizada ha sido la 2.3.1.

### 4.4.2. Dogs vs Cats

Este ha sido el primer dominio utilizado para realizar experimentación con redes de neuronas. La documentación del trabajo realizado se ha incluido en un anexo en el que se ha tratado con parte de la experimentación previa (véase ANEXO III: EXPERIMENTACIÓN PREVIA, C.1. Experimentación en el dominio de Dogs vs Cats). A continuación se expondrá un breve resumen.

El dominio de Dogs vs Cats [41] ha sido el primer dominio utilizado para realizar experimentación y aprender sobre redes de neuronas. Este dominio es muy común dentro del mundo de la visión artificial y consiste en clasificar imágenes que contienen un perro o un gato. Es considerado por muchos como el "*Hello world*" de las redes de neuronas.

La experimentación realizada en este dominio ha sido muy útil para conocer muchas de las técnicas que se pueden utilizar para obtener mejores modelos. Además, ha servido

para aprender a trabajar utilizando una metodología de experimentación concreta, especificada anteriormente. El concepto más importante aprendido fue la metodología experimental, de manera que sea posible comparar los experimentos y saber qué cambio ha sido el responsable de la mejora.

Un aspecto que ofrecen las redes convolucionales a diferencia de otras técnicas de aprendizaje automático es la posibilidad de visualizar las representaciones que aprenden. Se han aplicado dos de estas técnicas de visualización en este dominio.

### 4.4.3. Sign Language MNIST

Este es el segundo dominio utilizado para realizar experimentación con redes de neuronas. La documentación del trabajo realizado con este conjunto de datos se ha incluido en un anexo en el que se ha tratado con parte de la experimentación previa (véase ANEXO III: EXPERIMENTACIÓN PREVIA, C.2. Experimentación en el dominio de Sign Language MNIST). A continuación se expondrá un breve resumen.

El conjunto de datos de Sign Language MNIST [42] es un conjunto de Kaggle que está compuesto por imágenes del lenguaje de signos para las letras. Este conjunto de datos tiene el objetivo de ser similar al gran conocido MNIST, al igual que han hecho con otros como, por ejemplo, Fashion-MNIST [43].

Usar este conjunto de datos ha permitido realizar más experimentación para afianzar los conocimientos adquiridos. Además, con este conjunto se ha practicado también con el pre-procesado de datos. De nuevo se ha podido observar cómo puede mejorar un modelo al utilizar técnicas como *Data Augmentantion* o la regularización *Dropout*. En este caso se ha alcanzado un porcentaje de acierto de **97,32 %** en el conjunto de test.

Una de las principales aportaciones de este dominio está relacionada con el problema de generalización de las redes de neuronas. El modelo obtenido era muy dependiente del dominio. Sería necesario entrenar con conjuntos de datos con mayor variabilidad para mejorar la generalización de una red. Por ejemplo, es este caso sería necesario utilizar diferentes manos, diferentes fondos, diferentes colores…

### 4.4.4. MNIST usando *Autoencoders*

El MNIST es un conjunto de datos de dígitos manuscritos. Este conjunto está formado por 60.000 ejemplos de entrenamiento y 10.000 de test. Está compuesto por imágenes de tamaño 28x28 con 256 niveles de gris. Se suele utilizar en tareas de clasificación, es decir, a partir de la imagen con el dígito manuscrito, indicar qué digito es el que contiene la imagen. Sin embargo, en esta fase se ha utilizado para aprender a utilizar *Autoencoders*. El objetivo es reconstruir en la salida la imagen de entrada lo mejor posible.

En este dominio se ha realizado experimentación con *Stacked Autoencoders* y con *Denoising Stacked Autoencoders*. A continuación, se mostrarán los experimentos y conclusiones extraídas más relevantes. Se evaluarán los resultados con ayuda de algunos ejemplos, de forma que se puedan obtener conclusiones visualmente. También se utilizará una métrica dada por la función de error MSE. Esta función compara la salida esperada con la salida obtenida, el objetivo es minimizarla para que ambas se parezcan lo más posible.

#### *4.4.4.1. Stacked Autoencoders*

Usando este tipo de red se han realizado algunos experimentos iniciales que han permitido comprender el funcionamiento de los *Autoencoders*.

El primer experimento, 0027-MNIST_ED-CCFDDRCtCt-0001 (MNIST_ED: MNIST_Encoder-Decoder), utiliza dos capas convolucionales de 32 y 64 filtros en el codificador, un vector latente de 16 valores, y dos capas convolucionales transpuestas de 64 y 32 filtros para recuperar el tamaño de la imagen. En este experimento se busca visualizar que todo se está realizando de forma correcta y ver el potencial de dicha red, por lo que se entrena durante únicamente un ciclo. Se obtienen resultados positivos, aunque no muy buenos.

En el siguiente experimento, 0028-MNIST_ED-CCFDDRCtCt-0002, se aumenta el número de ciclos de entrenamiento de 1 a 10. En este caso se observan buenas reconstrucciones (véase Figura 4.3). La función MSE obtiene un error de 0,007, que servirá para compararlo con otros experimentos.

Por último, se aumenta el tamaño del vector latente para observar si la red es capaz de reconstruir los caracteres con mayor detalle. Este cambio se realiza en el experimento 0029-MNIST_ED-CCFDDRCtCt-0003.

Al aumentar el tamaño del vector latente, la red ha sido capaz de representar en dicho vector una mayor cantidad de detalles de los datos, logrando reconstrucciones con un mayor detalle (véase Figura 4.4). En este caso se obtiene un valor de 0,0005 de error en la función MSE, mucho menor que en el experimento anterior.

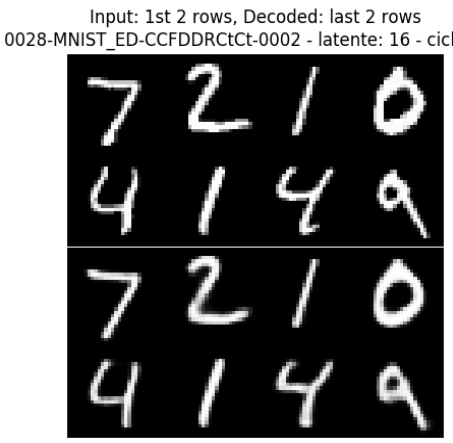



Figura 4.3: Experimento 0028-MNIST_ED-CCFDDRCtCt-0002.

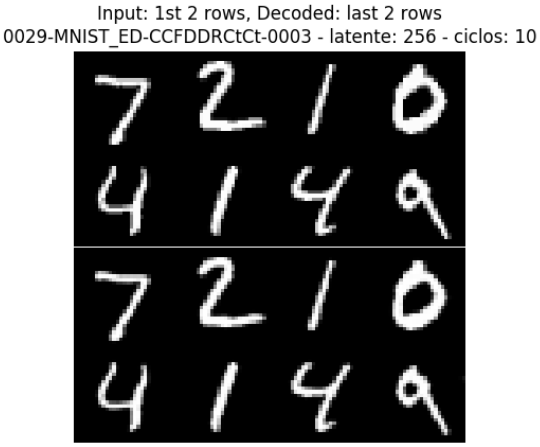



Figura 4.4: Experimento 0029-MNIST_ED-CCFDDRCtCt-0003.

*Las dos primeras filas son 8 imágenes de entrada. Las dos filas inferiores son las 8 reconstrucciones dadas por la red.*
*El experimento 0029 (error de 0,007) obtiene mejores resultados que el 0028 (error de 0,0005). Esto se puede observar en las imágenes, por ejemplo, la reconstrucción del dígito '4' situado en la cuarta fila y tercera columna tiene un detalle reconstruido en su parte inferior.*

#### *4.4.4.2. Denoising Stacked Autoencoders*

Tal y como se ha visto en el estado del arte, la introducción de ruido en las entradas crea redes más robustas y con mayor capacidad de generalización. En este dominio será complicado visualizar dicho efecto ya que ya se están alcanzando muy buenos resultados.

Para añadir ruido sobre las entradas se utiliza una capa de Keras llamada GaussianNoise a la que se le indica la desviación típica de una distribución Gaussiana centrada en cero. Esta capa se coloca al inicio y añade ruido a la entrada, haciendo así que cada imagen que entra a la red sea diferente al resto. La cantidad de ruido es un factor importante, ya que dependiendo de ello podría ser beneficioso o perjudicar.

En el experimento 0036-MNIST_ANoise-CCCFDDRCtCtCt-0003 se añade una pequeña cantidad de ruido, de 0,1 de desviación típica. Se puede observar en la Figura 4.5 como la red ha conseguido eliminar el ruido introducido. En este experimento se obtiene un error de 0,006 mediante la función MSE.

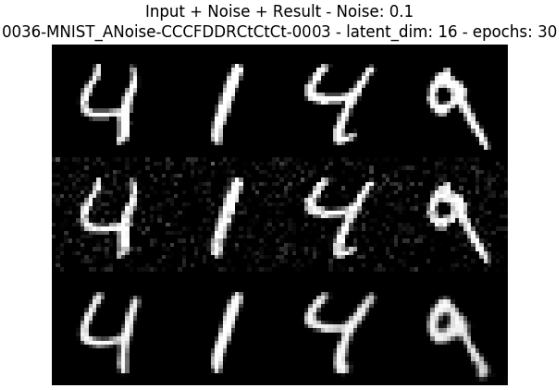



Figura 4.5: Experimento 0036-MNIST_ANoise-CCCFDDRCtCtCt-0003.

*Primera fila: Imagen original; Segunda fila: Imagen original con una pequeña cantidad de ruido: Tercera fila: Reconstrucción de la imagen dada por la red. Error dado por MSE: 0,0066.*

En el experimento 0037-MNIST_ANoise-CCCFDDRCtCtCt-0004 se aumenta la desviación típica de la distribución a 0,5, haciendo que el ruido sea más notable.

Algo que sí se puede observar en los resultados de este experimento (Figura 4.6) es como la red no reconstruye tanto algunos detalles, como el "rabito" que tiene el dígito '4' mostrado. Esto podría estar ocurriendo porque la red está aprendiendo características generales del dígito, reconstruyendo aquello imprescindible y sin llegar a detalles exclusivos de algunos ejemplos. Se podría decir que la red está generalizando. En este caso se obtiene un error de 0,0113, superior al obtenido en el experimento obtenido por la menor reconstrucción de los detalles mencionados.

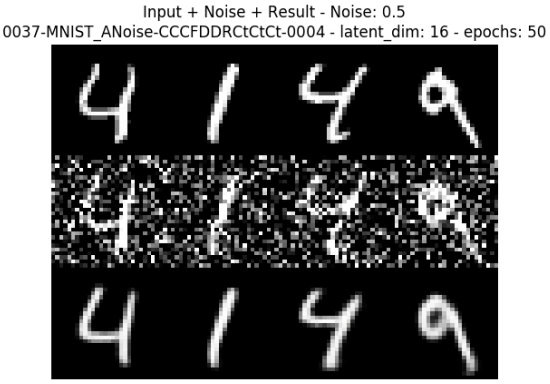



Figura 4.6: Experimento 0037-MNIST_ANoise-CCCFDDRCtCtCt-0004.

En la Figura 4.7 se puede observar un caso con exceso de ruido, en el que las reconstrucciones no son correctas. En ese caso, el ruido ha empeorado el proceso de entrenamiento ya que los datos de entrada han sido muy afectados. Se obtiene un error de 0,024, bastante superior a los casos anteriores.

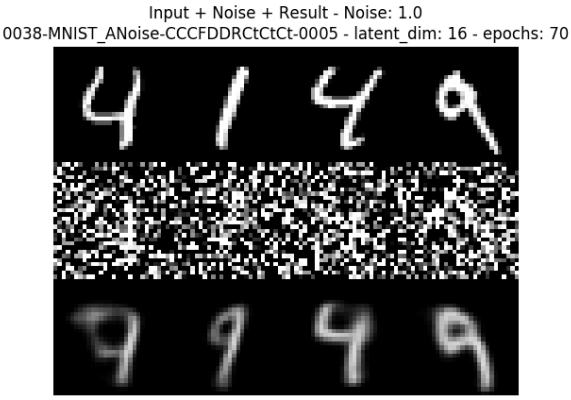



Figura 4.7: Exceso de ruido. MNIST. Experimento 0038.

Los *Denoising Autoencoders* consiguen una mejor representación de los datos en el vector latente. Aunque en este caso la mejora no sea apreciable visualmente, esto es debido a la sencillez del dominio. Más adelante se podrá observar como este tipo de red ha sido de vital importancia para el proyecto.

Se ha realizado una prueba en este dominio que ha permitido comprender algo más el funcionamiento del espacio latente creado por un *Autoencoder*. La prueba consiste en utilizar como entrada dos dígitos superpuestos, de forma que la reconstrucción esperada sea uno de los dos dígitos o una combinación de ambos. El resultado obtenido es una interpolación entre ambos dígitos (véase Figura 4.8)

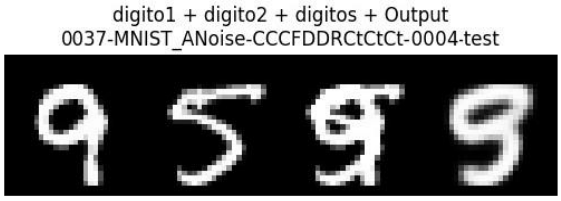



Figura 4.8: Prueba de reconstrucción con dos dígitos en la entrada.

*La entrada para este modelo es la tercera imagen mostrada, la combinación de ambos dígitos. La reconstrucción obtenida es la cuarta imagen. Como reconstrucción se obtiene una especie de interpolación entre ambos dígitos.*

El espacio latente contiene una representación comprimida de la imagen, de forma que el decodificador pueda utilizar esa información para realizar las reconstrucciones. En caso de utilizar un vector latente de dos valores, se podría representar el espacio latente como se puede ver en la Figura 4.9. El decodificador lee dos valores del vector latente y consulta la región del espacio correspondiente. En la entrada de la prueba realizada, la imagen tiene características tanto del ‘9’ como del ‘5’, por lo que el vector latente contiene valores correspondientes a una reconstrucción entre ambos, es decir, una interpolación entre ambas reconstrucciones.

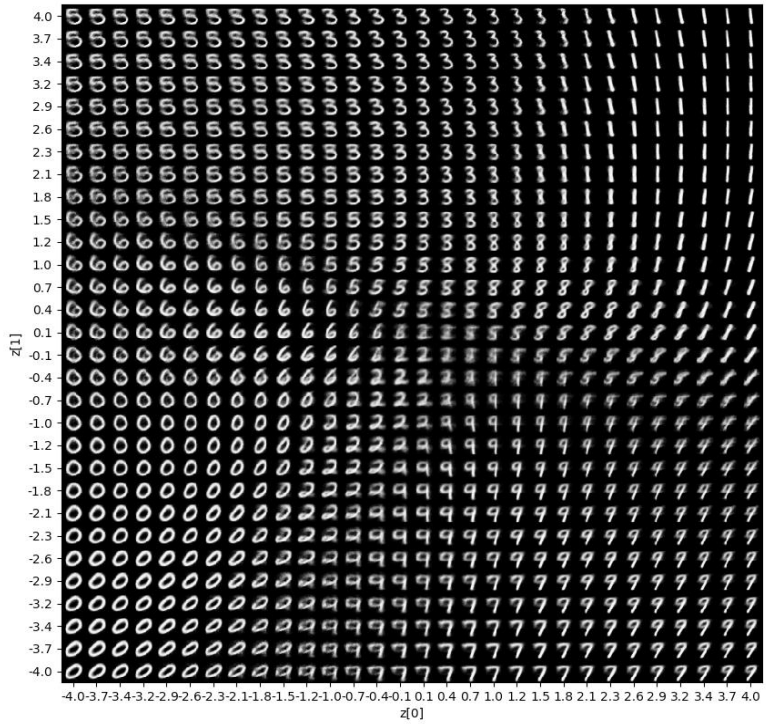


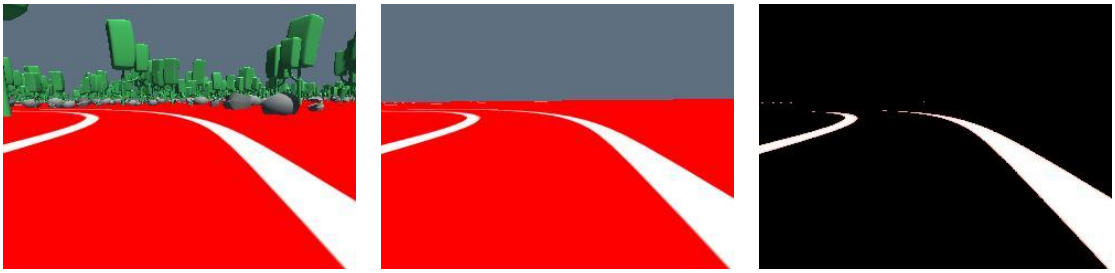

Figura 4.9: Representación del espacio latente de dimensión dos. Fuente: [23]

### 4.4.5. Primera Aproximación a la tarea de extracción de líneas de la carretera

En este apartado se mostrará la experimentación realizada con un conjunto de datos creado en un entorno de Unity de un TFG en realización. El objetivo de esta experimentación es comprobar la viabilidad de una de las tareas en las que se basa la idea principal del proyecto y practicar la forma de abordarla.

La tarea que se quiere realizar consiste en utilizar un *Autoencoder* para construir una red que aísle las líneas de la carretera del resto de elementos.

El conjunto de datos utilizado se compone de un total de 1.722 imágenes de un entorno simulado en Unity. Las imágenes se toman en un entorno compuesto por un par de líneas que delimitan una carretera, y piedras y árboles de fondo que componen el paisaje. Además de esas imágenes, se dispone de las mismas pero con el paisaje eliminado (véase Figura 4.10, Imagen a y b). El tamaño de las imágenes es de 320x240.



(a) Imagen con paisaje (b) Imagen sin paisaje (c) Imagen solo con líneas

Figura 4.10: Imágenes del conjunto de datos de Unity.

En la entrada de la red se introducen las imágenes con paisaje (Figura 4.10, Imagen (a)). En la salida esperada se introducen solo las líneas de la carretera de esas mismas imágenes (Figura 4.10, Imagen (c)).

La imagen que solo contiene líneas se obtiene a partir de la imagen sin paisaje (Imagen (b) en la Figura 4.10). Para obtenerla se aplica una máscara mediante OpenCV para eliminar los colores y mantener únicamente el color blanco de las líneas. En este caso los colores son totalmente sólidos y es una tarea sencilla. Sin embargo, cuando se realice en un entorno artificial, aunque los colores sean sólidos, será una tarea más complicada.

A partir de dicho entrenamiento no supervisado, el objetivo es obtener una red que realice la siguiente tarea: a partir de una imagen de la escena (carretera con paisaje), el *Autoencoder* reconstruye en la salida únicamente las líneas de la carretera.

A continuación, se mostrarán los experimentos más relevantes realizados. Ya que es una tarea muy visual, se aportarán imágenes y referencias a vídeos que muestren los resultados obtenidos.

#### 4.4.5.1. Experimentación

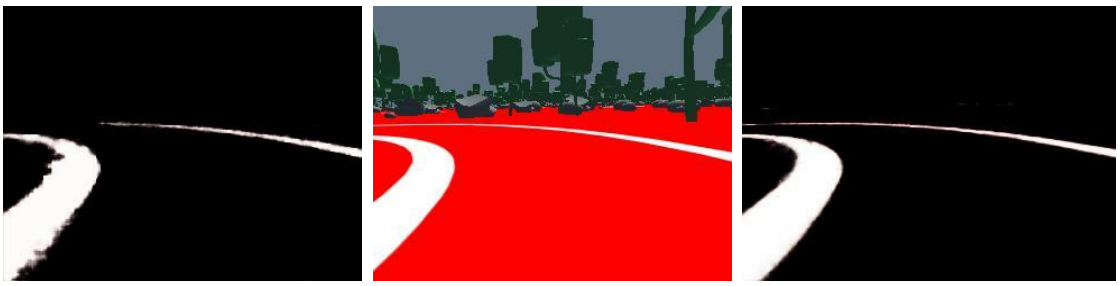
El conjunto de datos de entrenamiento se crea en ficheros del tipo NPZ[3]. Se crean dos ficheros, uno para entrenar con el 70% del total y otro para hacer test con el resto. Los conjuntos están formados por la entrada (imagen con paisaje) y lo que se espera en la salida de la red (imagen solo con las líneas). Volviendo a la Figura 4.10, se puede ver un ejemplo de estos datos, siendo la Imagen (a) la entrada y la Imagen (c) la salida esperada.

El primer experimento relevante realizado es el *0042-simulador-CCCFDDRCtCtCt-0002*. En este experimento se utilizan tres capas convolucionales, un vector latente de 16 valores y 20 ciclos de entrenamiento. Se consigue realizar la reconstrucción, pero no es del todo buena. Se puede ver la reconstrucción de un *frame* en la Figura 4.11, Imagen (a) y la secuencia completa en el **Vídeo 2.2**.

En el siguiente experimento, *0043-simulador-CCCCFDDRCtCtCtCt-0003*, se añade una capa convolucional. Al aumentar el tamaño de la red, ha sido necesario aumentar el número de ciclos de entrenamiento a 40. En este caso se consiguen resultados más nítidos y con menor ruido. Se puede ver la reconstrucción de un *frame* en la Figura 4.11, Imagen (c) y la secuencia completa en el **Vídeo 2.3**.

En el experimento *0044-simulador-CCCCFDDRCtCtCtCt-0004*, se aumenta el tamaño del vector latente a 32 para observar si se alcanza alguna mejora. Observando las reconstrucciones por separado no se observa mejora con respecto al 0043. Aunque si se visualiza de forma dinámica (véase **Vídeo 2.4**) se puede ver como las líneas se reconstruyen en posiciones más precisas (se observa mayor movimiento). No se muestra un ejemplo de reconstrucción en una figura dado que los resultados son muy parecidos a los del experimento 0043.

[3] NPZ es un formato utilizado por la librería de *NumPy* para guardar de forma eficiente un array en un fichero comprimido.

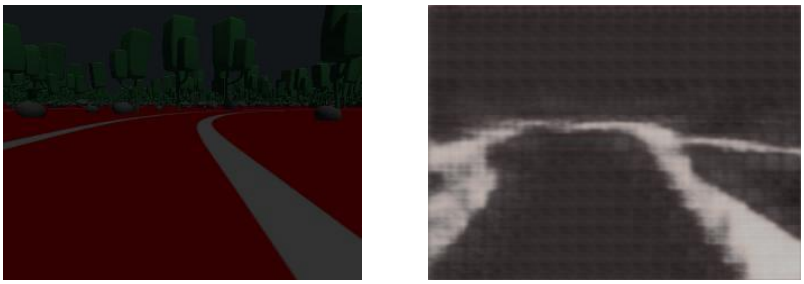


(a) Reconstrucción en el experimento 0042 (b) Imagen original (c) Reconstrucción en el experimento 0043

Figura 4.11: Comparación de reconstrucciones de las líneas de la carretera.

Una de las pruebas realizadas ha consistido en introducir las mismas imágenes de entrada pero modificando su cantidad de iluminación. Teniendo un rango de 0 a 2, donde 1 es la iluminación original, se han modificado las imágenes estableciendo la iluminación en 0,2, 0,6, 1,4 y 1,8. Cuando el valor está por debajo de 1, se oscurece la imagen y cuando está por encima de 1, aumenta su iluminación.

El objetivo de esta prueba consiste en comprobar la robustez de los modelos creados. Esta prueba se ha realizado utilizando el modelo obtenido en el experimento 0044. Se obtienen algunas reconstrucciones incorrectas, principalmente cuando la iluminación es baja (Figura 4.12 y **Vídeo 2.5**).



(a) Imagen con iluminación modificada (b) Reconstrucción obtenida

Figura 4.12: Reconstrucción deteriorada a partir de una imagen con iluminación y contraste bajos.

Para estudiar el problema se realiza el experimento *0045-simuladorBrillos-CCCCFDDRCtCtCtCt-0001*. El entrenamiento se realiza con las mismas imágenes que antes pero variando su cantidad de iluminación como se ha indicado. En este caso, al haber entrenado con modificaciones de iluminación, la red sí que consigue reconstruirlas satisfactoriamente (véase **Vídeo 2.6**). Como conclusión, la iluminación debe ser un factor a tener en cuenta si es necesario que la red sea robusta ante estas situaciones. De cara a la experimentación en un entorno real, las imágenes obtenidas ya podrían tener algo de variabilidad en su iluminación, como sombras o zonas a las que llega menos cantidad de luz, por lo que esto podría no ser un problema.

Aunque se han obtenido resultados positivos en los experimentos, en el siguiente apartado se modifica el conjunto de datos para intentar complicarlo y simular las imágenes que se podrían obtener en un entorno real.

### 4.4.6. Segunda Aproximación a la tarea de extracción de líneas de la carretera

En esta segunda aproximación se modifica el conjunto de datos para complicar algo más la tarea. Se sigue utilizando el conjunto de datos anterior como base del que se va a utilizar en este caso.

El paisaje de las imágenes obtenidas en el apartado anterior es siempre el mismo, además de ser muy sólido. Siendo así podría ser muy sencillo ignorarlo en la reconstrucción. Para complicar la tarea se utiliza una clase de un conjunto de datos llamado *Places*, creado por el MIT [44]. Este conjunto de datos tiene una gran cantidad de imágenes para realizar reconocimiento de escenas. Se utiliza la clase *forest_path* que contiene 15.100 imágenes de paisajes.

La modificación del conjunto de datos consiste en eliminar los colores del fondo de la imagen quedando solo las líneas de la carretera. Se superpone ese resultado sobre las imágenes de paisajes. El objetivo es comprobar si la red aprende a extraer las líneas independientemente del entorno en el que se encuentran.

Ya que se dispone de una cantidad menor de imágenes de líneas de carretera (1.722) que de paisajes (15.100), se repiten las de líneas hasta alcanzar el total de las segundas.

(a) Ejemplo 1 (b) Ejemplo 2 (c) Ejemplo 3

Figura 4.13: Ejemplos del dominio de Unity modificado.

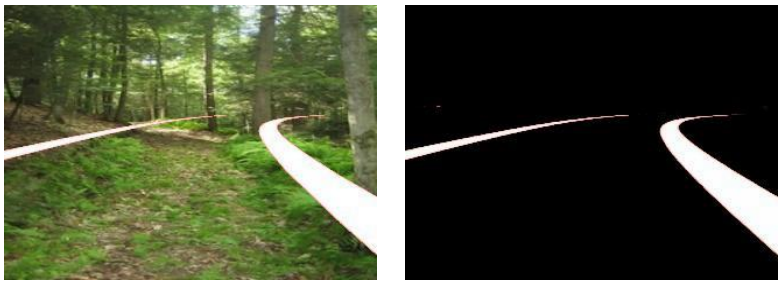
*Se toman las imágenes del dominio de Unity sin paisaje y se eliminan los colores sólidos, quedando únicamente las líneas de la carretera. Se superpone el resultado sobre una imagen de un paisaje.*

#### 4.4.6.1. Experimentación

Se utiliza el 70% de los paisajes para entrenar y el resto para realizar el test. Además, al aumentar el tamaño del conjunto de datos este no puede ser guardado en un fichero NPZ como antes. En este caso, se guardan directamente las imágenes en una carpeta y se alimenta el entrenamiento de la red mediante generadores.

Otro de los cambios importantes al aumentar el tamaño del conjunto de datos ha sido el tiempo de entrenamiento. Mientras que en el experimento *0042-simulador-CCCFDDRCtCtCt-0002* cada ciclo tardaba unos 27 segundos, al aumentar el conjunto de datos este tiempo ha pasado a unos 270 segundos. Como el tiempo total de entrenamiento sería muy alto, se decide utilizar las dos GPU del ordenador para entrenar los modelos. Para ello, se utiliza un pequeño programa que indica a Tensorflow que puede utilizar dos dispositivos (Multi GPU) y reestructura la red para que sea posible entrenarla de esta forma. De esta forma se ha conseguido reducir el tiempo de entrenamiento por ciclo a 175 segundos.

El primer experimento que usa este nuevo conjunto de datos es el *0046multiGPU-simuladorPaisajes-CCCCFDDRCtCtCtCt-0001*. De nuevo, se consigue reconstruir las líneas de la carretera de forma correcta. El resultado mostrado en el **Vídeo 2.7** está compuesto por paisajes no vistos anteriormente por la red (conjunto de test), consiguiendo realizar la tarea independientemente del fondo. Se puede observar un ejemplo de este experimento en la Figura 4.14.



(a) Imagen de entrada (b) Reconstrucción

Figura 4.14: Resultados del experimento 0046.

Para llevar algo más al límite el *Autoencoder* se genera un conjunto de datos con algo más de variabilidad. Para ello se aplica *Data Augmentation* sobre las líneas de la carretera. Cada imagen se rota -15 y +15 grados y se voltean horizontalmente, consiguiendo multiplicar por 6 las imágenes de líneas de carretera. Se podría haber aumentado aún más el conjunto, pero esto complicaba bastante la tarea de preprocesamiento de datos. Se pueden observar algunos ejemplos en la Figura 4.15.

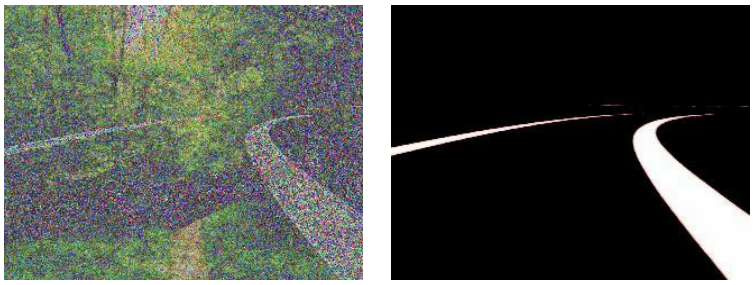
(a) Rotación negativa (b) Original (c) Rotación positiva

Figura 4.15: Ejemplos de *Data Augmentation* sobre las líneas de la carretera.

El experimento que prueba este conjunto de datos es el *0047multiGPU-simuladorPaisajes-CCCCFDDRCtCtCtCt-0002*. De nuevo, se obtienen resultados positivos (véase **Vídeo 2.8**). Las líneas siguen siendo muy artificiales por lo que se intenta complicar algo más la tarea.

En la siguiente modificación al conjunto de datos se aplica un programa de aumentación de conjuntos de datos precisamente para conducción autónoma. Esta librería se encuentra en GitHub y tiene el nombre de Automold [45]. Las modificaciones aplicadas a las imágenes son variaciones de brillo, adición de sombras, nieve, lluvia, niebla, gravilla, reflejos del sol, efecto de velocidad… etc. Además, se aplica ruido sobre la imagen final para complicar aún más la tarea.

El experimento que prueba este conjunto de datos es el *0052multiGPU-simuladorPaisajesAugMIT2-CCCCFDDRCtCtCtCt-0001*. En este caso algunas reconstrucciones se hacen algo peor, es comprensible ya que la imagen de entrada está muy dañada. Se puede ver un ejemplo de reconstrucción en la Figura 4.16 y la secuencia completa en el **Vídeo 2.9**.



(a) Imagen de entrada (b) Reconstrucción

Figura 4.16: Resultados del experimento 0052.

Aun complicando el conjunto de datos se obtienen resultados positivos, éstos se podrían estar dando por utilizar un conjunto de datos sencillo ya que provienen de una simulación. Aunque se han intentado añadir muchas complicaciones, al fin y al cabo las líneas de carretera siguen siendo son las mismas y eso podría estar facilitando la extracción.

Finalizando con esta experimentación, se ha observado que la tarea que se quería realizar es viable, al menos, en un simulador. Aunque se ha intentado complicar este conjunto de datos, no se ha conseguido algo tan complejo como lo que se buscaba. Asemejarlo más a la realidad mediante un simulador podría ser una tarea demasiado difícil, por lo que se concluye con este dominio y se comenzará con la experimentación en un entorno real en la fase IV.

## 4.5. Fase III: Sistema de control y obtención de datos de entrenamiento

En esta fase se trata con los aspectos técnicos necesarios para desarrollar un sistema de control del vehículo a escala y obtención de datos de entrenamiento.

Se hará referencia a varios vídeos en los que se puede visualizar mejor los aspectos que se mencionan. Como ya se ha comentado anteriormente, los vídeos se encuentran organizados por fases en un repositorio.

El sistema que se quiere desarrollar realizará varias tareas:

- Crear una conexión entre un ordenador y el vehículo.
- Mostrar en el ordenador un vídeo sobre lo que captura el vehículo en tiempo real.
- Controlar el vehículo mediante un dispositivo conectado a un ordenador.
- Guardar las imágenes del vídeo en tiempo real y las etiquetas de dirección y velocidad que permitirán formar conjuntos de entrenamiento.

El sistema de control y obtención de datos completo será descrito después de tratar con todo lo necesario. A continuación, se describirá cada uno de esos aspectos y las decisiones tomadas para cada uno de ellos.

### 4.5.1. Dirección y velocidad del vehículo

En este apartado se trata con aspectos mecánicos del vehículo que permiten que se mueva de forma adecuada, la dirección y la velocidad.

Será necesario que el vehículo pueda realizar giros amplios para desarrollar un circuito de un tamaño razonable. También es necesario que sea capaz de mantener una velocidad baja y constante. Esto último podría ser necesario más adelante para que el SBC tenga tiempo suficiente de hacer inferencia sin que el vehículo avance mucha distancia.

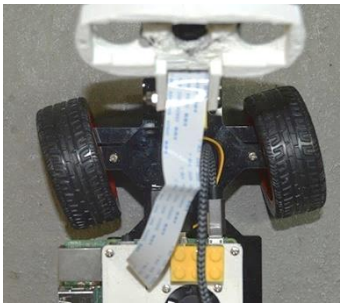
En primer lugar se ha evaluado la capacidad de giro del vehículo. Para ello se ha medido el radio de la circunferencia creada por el vehículo al dar vueltas sobre sí mismo a una velocidad reducida. Tal como se indica en el anexo sobre señales PWM (véase ANEXO I: SEÑALES PWM), el giro del vehículo viene determinado por dos valores PWM que indican la posición máxima a la izquierda y derecha dada por un servomotor. Modificando estos valores se pueden alcanzar diferentes capacidades de giro (véase Tabla 4.5).

La primera fila de la Tabla 4.5 indica los valores por defecto en el vehículo. Sin embargo, experimentando con otros valores se ha alcanzado un radio de giro mayor, que permite realizar curvas más cerradas. Se han seleccionado los valores (350, 550) como valores límite de giro izquierda y derecha respectivamente. Se puede observar la diferencia entre (400, 500) y (350, 550) en la Figura 4.17 y en un vídeo al que se hará referencia a continuación.

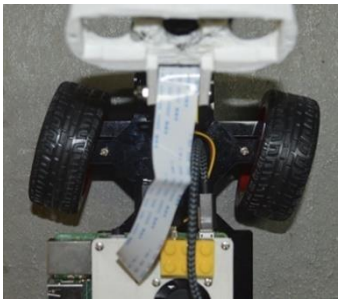
Tabla 4.5: Relación entre valores PWM del servomotor de dirección y radio de giro.

| **Valor PWM máximo del servomotor de dirección** | | |
|---|---|---|
| **Izquierda** | **Derecha** | **Radio de giro** |
| 400 | 500 | 41cm |
| 375 | 525 | 26cm |
| 350 | 550 | 21cm |

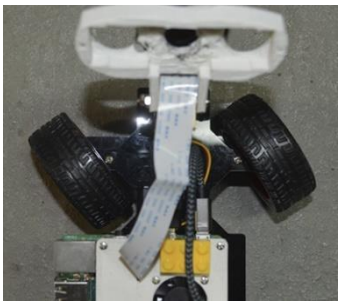
***Nota.*** *Los valores PWM de esta tabla indican la posición que toma el servomotor para que el eje de dirección del vehículo quede orientado hacia la dirección mencionada en la tabla. Entre el valor a izquierda y derecha existen valores que permiten establecer la dirección hacia un punto intermedio.*



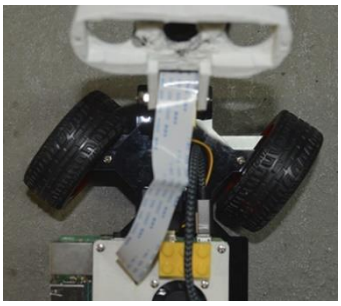
(a) Valor PWM 400 (izq.)



(b) Valor PWM 500 (dcha.)



(c) Valor PWM 350 (izq.)



(d) Valor PWM 550 (dcha.)

Figura 4.17: Comparativa entre valores PWM para el servomotor de giro.

*El valor PWM indicado en la anotación de cada imagen indica al servomotor la posición en la que se tiene que colocar.*

En el **Vídeo 3.1** se pueden visualizar las diferencias en la conducción con los valores (400, 500) (radio de 41 cm) y los valores (350, 550) (radio de 21cm). Utilizando estos

últimos valores se puede observar como la capacidad de maniobra es mucho mayor. Una de las grandes ventajas que esto implica es la posibilidad de crear circuitos de un tamaño más reducido y razonable.

En segundo lugar se ha evaluado la velocidad. Para ello se ha medido el tiempo que el vehículo tarda en recorrer una distancia de unos 3 metros para calcular la velocidad en RPM del vehículo a diferentes velocidades. Al igual que con los valores de giro, la velocidad del vehículo también viene determinada por valores PWM. El rango de valores de velocidad es [0, 4.000].

Tabla 4.6: Relación entre valores PWM en los motores de tracción y velocidad.

| | **Valor PWM** | **Velocidad (RPM)** | **Velocidad (m/s)** | **Velocidad (km/h) a escala real (1:16)** |
|---|---|---|---|---|
| **Velocidad mínima** | 800 | 79 RPM | 0,27 m/s | 15 km/h |
| **Velocidad máxima** | 4.000 | 230 RPM | 0,78 m/s | 45 km/h |

***Nota.*** *La velocidad en metros por segundo se ha calculado teniendo en cuenta que las ruedas del vehículo tienen un perímetro de 20,5 centímetros. Los motores utilizados son los TT Motor DC 1:48.*

La velocidad mínima ha sido establecida experimentando con diferentes valores que permiten que el vehículo se mueva sin que se quede parado por falta de torque. El valor PWM que mejor resultado daba ha sido 800, haciendo que las ruedas del vehículo giren a 79 RPM y proporcionando una velocidad de 0,27 m/s.

La velocidad mínima alcanzada es aceptable, aunque se podría tener algún problema futuro. Si el tiempo de inferencia en la tarea de conducción fuese elevado, sería necesario que el vehículo se desplace a menor velocidad. Esto requeriría disponer de motores con un mayor torque que permitan mover el vehículo a velocidades menores. Se estudiarán diferentes motores en la última fase.

### *4.5.2.* Dispositivo de control

El control del vehículo a través de un teclado podría ser complicado, por lo que es necesario utilizar algún dispositivo que facilite la conducción. En este apartado se describirá la elección de un dispositivo adecuado y las tareas realizadas para desarrollar el control del vehículo.

Pygame es un conjunto de módulos del lenguaje Python que permite la creación de videojuegos en dos dimensiones de manera sencilla. Uno de los módulos está destinado a la monitorización de la entrada (como un teclado, un ratón, un *joystick*… etc). Se utilizará este módulo para programar un sistema de control del vehículo.

#### 4.5.2.1. Control con un mando de consola

Normalmente el control mediante el módulo de Pygame se aborda utilizando una cola de eventos. Estos eventos muestran la tecla pulsada o su valor en caso de poder tenerlo (ejes).

En una primera aproximación al control, se conecta mediante *bluetooth* un mando de Xbox One (véase Figura 4.18) directamente a la Raspberry. En el SBC se ejecuta el programa de control que será descrito a continuación.


Figura 4.18: Control del vehículo usando el mando de Xbox One.

El control básico consiste en leer una cola de eventos y tratarlos. En este caso se tiene en cuenta la aceleración, tanto hacia delante como hacia atrás (para que sea posible maniobrar) y el control de dirección.

Los valores de aceleración y giro obtenidos están en el rango [-1, 1], que son mapeados al rango [0, 4.000] (valores PWM de los motores), y a los valores PWM de dirección establecidos.

La conexión directa mediante *bluetooth* con el vehículo tiene un alcance muy limitado. Ha sido necesario conectar el mando a un ordenador, ya que está situado en un punto fijo, y transmitir la información de control a través de una conexión wifi.

Se crea una red privada en la que únicamente se conecta el ordenador y el vehículo para evitar problemas. Se ejecuta el programa de lectura de eventos del dispositivo de control en el ordenador y estos se envían mediante comandos al vehículo.

La comunicación entre ambos dispositivos se puede realizar a través de una conexión del tipo TCP o UDP.

- TCP: Este protocolo está orientado a conexión, es decir, es necesario establecer una conexión entre ambos dispositivos antes de enviar los datos. Verifica el envío de los datos, siendo altamente confiable.
- UDP: Este protocolo no está orientado a conexión, envía los datos directamente. No verifica el envío de datos, por lo que se pueden perder algunos paquetes.

El protocolo UDP es más rápido ya que no realiza tareas de control de los datos enviados. En la tarea que se quiere realizar, la velocidad de transmisión es muy importante para reducir la latencia, por lo que se decide utilizar UDP. Además, la pérdida de algún paquete no causaría ningún problema dado que el control es interactivo y se actualiza constantemente.

En el **Vídeo 3.2** se puede observar cómo funciona el control del vehículo utilizando este mando.

Aunque conducir con este tipo de dispositivo es totalmente viable, genera algunos comportamientos no realistas. La posibilidad de cambiar de dirección con un solo movimiento de pulgar produce movimientos muy bruscos. Con el objetivo de conseguir una conducción más realista se tomó la decisión de adquirir un volante de conducción.

#### 4.5.2.2. Control con un volante de conducción

Se utiliza el kit de conducción Logitech G29 Driving Force, que incluye volante y pedales (véase Figura 4.19). Para controlar el vehículo con este kit se utiliza el mismo programa utilizado anteriormente con alguna modificación. También se utiliza una

aplicación llamada Oversteer que permite configurar el volante (se configuran principalmente dos aspectos, limitación de los grados de giro del volante y fuerza de auto centrado).


Figura 4.19: Control del vehículo usando el volante Logitech G29.

En primer lugar, el valor de aceleración dado por los pedales está en el rango [1, -1], al contrario que en el mando. Esto se soluciona multiplicando por -1 el dato de control.

En segundo lugar, ya no se utiliza una cola de eventos. Este dispositivo de control genera una cantidad de eventos mucho mayor que el mando, tal vez por tener una mayor frecuencia de actualización. Por ello, en lugar de atender a todos los eventos ocurridos únicamente se toma el último evento capturado. Cada vez que se captura un evento, el programa duerme durante 0,01 segundos, que corresponde con una captura de 100 eventos por segundo.

Se añade una funcionalidad al programa. Pulsando uno de los botones del volante se activa la velocidad mínima en el vehículo, de forma que este se mueve a una velocidad constante y solo es necesario asumir el control de la dirección. Esta funcionalidad será útil para capturar datos de entrenamiento conduciendo a una velocidad constante.

Este ha sido el control escogido para realizar la conducción, se hará referencia a un vídeo sobre ello en el último apartado de esta fase.

### 4.5.3. Determinación de FPS necesarios para la conducción

Ya que los recursos son limitados (procesamiento del SBC o calidad de la conexión wifi), la cantidad de *frames* por segundo recibidos del vídeo en tiempo real podría ser baja. Si esto ocurre, no será posible conducir con precisión.

Para conocer los FPS mínimos necesarios, se ha realizado una conducción en un simulador experimentando con diferentes valores. Esta se ha recogido en el **Vídeo 3.3**, en el que se muestran secuencias de conducción a 15, 10 y 5 FPS a una velocidad reducida. Además, se realiza con el volante que se utilizará para controlar el vehículo a escala reducida.

Un aspecto claro es que cuanto mayor sea la cantidad de FPS, mayor es la precisión en la conducción a velocidades mayores. Sin embargo, se establece un límite inferior de 10 FPS para conducir de forma supervisada, por debajo de esa cantidad la tarea de conducción manual aun con una velocidad reducida se complica bastante.

### 4.5.4. Cámara del vehículo

Tal como se indicó en el análisis del problema, se va a utilizar una Pi Camera como cámara del vehículo. Sin embargo, la que se utilizó inicialmente no tenía suficiente ángulo de visión como para abarcar las líneas de la carretera en curvas y otras situaciones. Por este motivo se comienza a utilizar una Pi Camera con una lente de gran angular, aumentando bastante el campo de visión.

En la Figura 4.20 se muestra una diferencia entre utilizar una lente normal y una de gran angular.

(a) Lente normal (Pi Camera rev 1.3) (b) Lente gran angular (Pi Camera rev 2.2)

Figura 4.20: Comparación entre lente estándar y lente gran angular.

*Se han colocado dos tiras de papel para marcar el campo de visión de la lente normal (imagen a). En la imagen b se han mantenido esas marcas para ver la diferencia. Ambas imágenes se han tomado desde la misma posición y siguiendo el mismo procedimiento.*

En el **Vídeo 3.4** se muestra el punto de vista del vehículo en un recorrido utilizando ambas cámaras para poder apreciar la diferencia.

### 4.5.5. Vídeo en tiempo real

El vídeo en tiempo real o *streaming* es una parte importante y necesaria para el sistema por lo que se ha estudiado a fondo. Se proponen diferentes métodos que se pueden procesar en una Raspberry. A continuación se muestran los detalles de cada propuesta.

- **Raspivid + MPlayer**: Raspivid es un programa de Raspberry que sirve para grabar vídeo. Utilizando este programa se puede capturar vídeo y enviarlo a otro dispositivo mediante *netcat*. Los datos recibidos en destino se reproducen a través de un reproductor llamado MPlayer.
- **Raspivid + GStreamer***:* Este método también utiliza Raspivid para capturar, pero el envío de datos lo realiza a través de una librería multimedia llamada GStreamer. En destino se utiliza también esta librería para reproducir el vídeo.
- **MJPG Streamer***:* Es una aplicación de línea de comandos que copia *frames* JPEG de una entrada (como una cámara) a una salida (como un servidor HTTP). El vídeo en tiempo real se puede visualizar desde otro dispositivo conectado a la misma red mediante un navegador.

Hay varios aspectos a tener en cuenta para seleccionar uno de los métodos.

En primer lugar, es necesario que la latencia (tiempo transcurrido entre captura y reproducción) se minimice para reproducir las imágenes en tiempo real y poder realizar la conducción de forma correcta.

En segundo lugar, es necesario guardar las imágenes visualizadas y tal vez modificarlas, como reducir (para datos de entrenamiento) o ampliar su tamaño (para mejorar la visualización al conducir) o añadir información en ellas a modo de HUD (*Head-Up Display*, información mostrada en pantalla). Esto se quiere realizar haciendo uso de OpenCV en Python, por lo que será necesario que el método de *streaming* permita incorporarlo.

Por último, un factor también importante es que sea sencillo de utilizar y tenga buena compatibilidad con el sistema operativo de la Raspberry para evitar problemas.

Teniendo en cuenta los aspectos anteriores, se descarta la opción de Raspivid junto a GStreamer ya que la instalación de dichas librerías ha presentado muchos problemas de compatibilidad. Además, el resultado obtenido no ha sido suficientemente bueno en cuanto a latencia.

El método de Raspivid usando *netcat* y MPlayer ofrece buenos resultados, aunque al intentar procesarlo mediante OpenCV aumenta bastante la latencia. Este método queda delegado como segunda opción ya que se podría llegar a una solución y utilizarlo.

Finalmente, se selecciona el método de MJPG Streamer. Este método ofrece una facilidad de ejecución mayor que el resto además de una compatibilidad inmediata. La latencia de este método es baja (alrededor de 0,1 segundos). En la Figura 4.21 se representa la diferencia de tiempo entre captura y reproducción. Una de las grandes ventajas de este método es que la Pi Camera captura los *frames* en JPEG y los envía directamente, sin necesidad de codificarlos en otro formato. Esta funcionalidad provoca que el consumo en CPU por la captura y transmisión de vídeo sea muy bajo (2-4% de uso de CPU).

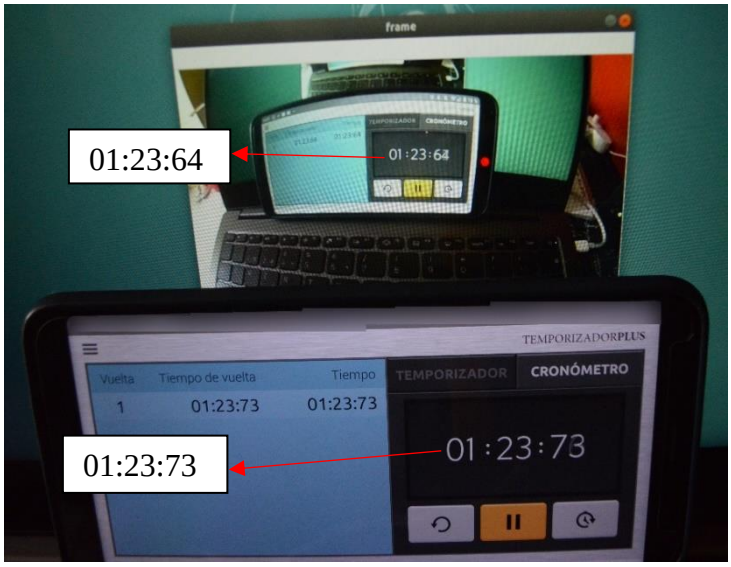



Figura 4.21: Cálculo de la latencia de MJPG Streamer.

*Utiliza una resolución de 640x480, recibe 30 FPS y transmite directamente lo que captura, sin codificar en otro formato. La cámara de transmisión captura un cronómetro, tomando una foto de la escena se puede ver la diferencia entre captura y reproducción. Latencia: (01:23:73 – 01:23:64) = 0,09 segundos.*

La resolución del vídeo se configura en origen (Raspberry en este caso), aunque se podría modificar en destino. Se podría disminuir para guardar imágenes en un tamaño inferior para formar conjuntos de datos de entrenamiento. También podría aumentarse (perdiendo calidad) para mejorar la visualización al controlar el vehículo.

En cuanto a los frames por segundo, aún reduciendolo a los 10 FPS comentados anteriormente (apartado 4.5.3 Determinación de FPS necesarios para la conducción), la latencia no disminuye, por lo que se mantiene en 30 FPS.

El parámetro que sí podría afectar a la latencia es el bitrate de vídeo, que se ha establecido en 10Mbps. Con este valor es posible visualizar los frames JPEG sin pixelaciones o ruido hasta la resolución de 640x480. A partir de ese tamaño de imagen sería necesario aumentar el bitrate, provocando un aumento de la latencia.

Debido a que éste es el método utilizado en el sistema final, se hará referencia a un vídeo del funcionamiento junto al control con el volante en el siguiente apartado.

### 4.5.6. Sistema de control completo

Después de haber tratado todos los temas necesarios, ya es posible unificarlos y construir el sistema que se encargará de ofrecer el control, visualización y la obtención de datos de entrenamiento.

Para poner en marcha el sistema completo ha sido necesario ejecutar dos tareas a la vez: el control del vehículo y la visualización y captura del vídeo en tiempo real. Ya que para guardar los datos de entrenamiento es necesario cruzar datos de ambas tareas (datos de control e imagen capturada), ha sido necesario ejecutar ambas tareas en el mismo programa. Para ello, ha sido necesario elegir entre concurrencia con procesos o con hilos en Python.

En este caso la decisión ha sido sencilla. El programa de visualización necesita tener acceso a los datos generados por el programa de control, éste detalle es algo decisivo. Utilizando diferentes procesos, estos trabajarían en diferentes espacios de memoria, haciendo que compartir datos sea algo más complicado o costoso. Sin embargo, los hilos trabajan en el mismo espacio de memoria, simplificando el acceso de memoria necesario.

Solo ha sido necesario acceder a los datos de control para leer el último valor escrito, por lo que no ha sido necesario utilizar mecanismos de control de acceso como cerrojos o semáforos.

Se puede observar el funcionamiento del sistema completo en el **Vídeo 3.5**, en el que se muestra la conducción con volante y pedales y la obtención de los conjuntos de datos de entrenamiento.

#### 4.5.6.1. Procesamientos en el vehículo

En la Raspberry se ejecuta un programa que se encarga de crear dos hilos.

Uno de ellos inicia el servidor de vídeo en tiempo real, que se quedará enviando *frames* hasta ser detenido.

El otro hilo se encarga de iniciar el programa de control del vehículo. En este programa se ejecuta en primer lugar una función que crea un socket de comunicación UDP. El programa quedará a la espera de llegada de paquetes. Cuando reciba paquetes, obtendrá la información de control necesaria y la trasladará a los actuadores del vehículo (motores de tracción y servomotor de dirección).

#### 4.5.6.2. Procesamientos en el ordenador de control

En el ordenador de control se realizan más tareas que en el vehículo. En este también se ejecuta un programa que crea dos hilos. Antes de lanzar los dos hilos, se instancia un objeto de la clase que tiene el control del volante para que ambos hilos tengan acceso a él.

En el primer hilo se inicia la configuración del volante necesaria para enviar datos al socket UDP creado en el vehículo. Cuando se ha realizado esta configuración, se comienzan a capturar los eventos generados por el volante y pedales y estos se envían al vehículo.

En el segundo hilo se realiza todo lo relacionado con la visualización del vídeo en tiempo real del lado cliente. El vídeo en tiempo real se encuentra alojado en una dirección HTTP a la que se puede acceder desde cualquier dispositivo en red con el vehículo. Por lo que se inicia una captura de este vídeo mediante OpenCV.

```
cap = cv2.VideoCapture("http://" + <RPI_IP> + ":8080/?action=stream")
```

A partir de lo anterior se pueden obtener los *frames* mediante un bucle. Dentro de este bucle se realiza el procesamiento necesario: aumento de tamaño del *frame* capturado para visualizar, inserción de información de control en él y visualización de los *frames* en pantalla (véase Figura 4.22).

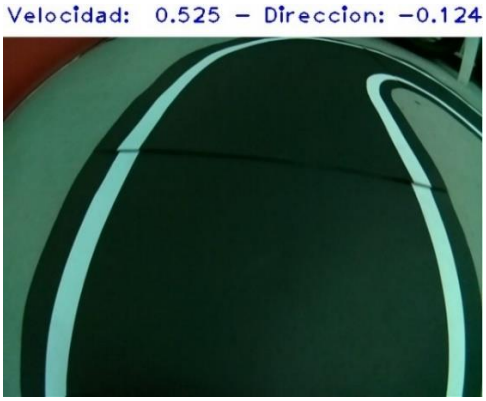



Figura 4.22: *Frame* capturado y procesado del vídeo en tiempo real.

Este sistema sería suficiente para conducir el vehículo. En caso de querer guardar datos para crear conjuntos de entrenamiento, sería necesario realizar alguna tarea adicional.

Para obtener los datos de entrenamiento ha sido necesario guardar los *frames* del vídeo en tiempo real y los datos de control. Para los datos de control se abre un fichero en el que se escribe en cada línea un identificador y el valor de giro y velocidad (obtenido mediante una llamada al objeto de la clase que tiene el control del volante). Los *frames* se guardan en una carpeta del ordenador usando como nombre el identificador anterior. Este identificador es el tiempo en milisegundos desde el 1 de enero de 1970.

Previa a la ejecución del programa de control se realiza la configuración del volante. Como ya se ha dicho anteriormente, esta configuración se realiza a través de un programa llamado Oversteer. La configuración utilizada para conducir el vehículo ha sido la siguiente: limitación de giro del volante a 360º y fuerza de auto centrado situada en un valor de 30 en un rango de 0 a 100. Estos valores se han obtenido mediante experimentación, buscando la configuración más adecuada para realizar la conducción.

Una vez realizado todo lo anterior, ya es posible conducir el vehículo y formar conjuntos de datos de entrenamiento.

## 4.6. Fase IV: Extracción de características de las líneas de la carretera (Primer entorno)

En este apartado se describirá la experimentación realizada con *Autoencoders* en el primer entorno real. Previa a esta experimentación se ha realizado una aproximación a

esta tarea en un entorno simulado (4.4.5. Primera Aproximación a la tarea de extracción de líneas de la carretera y 4.4.6. Segunda Aproximación a la tarea de extracción de líneas de la carretera).

En la aproximación mencionada, la tarea se consigue realizar satisfactoriamente. Dado que era un entorno simulado, la tarea era más sencilla, pero transformar el conjunto de datos en algo realista resultaba muy complejo.

Con el sistema de control y obtención de datos de entrenamiento terminado, se comienza con la nueva fase. La experimentación realizada en el entorno simulado ha permitido enfrentarse a problemas de forma incremental, facilitando en gran medida el trabajo.

La presente experimentación se realiza como primera aproximación en un entorno real. Por ello, el entorno de trabajo en esta fase es algo más sencillo que el que se podrá ver en la fase final.

El objetivo de la tarea que se quiere abordar consiste en entrenar un *Autoencoder* que sea capaz de ignorar lo accesorio de una escena, reconstruyendo únicamente las líneas de una carretera. Se obtendrá un vector de características (vector latente del *Autoencoder*) que representa las líneas de la carretera.

Para realizar esta tarea es necesario un escenario artificial mediante el que sea posible reproducir imágenes de escenarios reales. Se seleccionarán adecuadamente los materiales para poder aplicar la técnica *Chroma Key*[4] y generar conjuntos de datos con imágenes aumentadas. Sin este escenario no sería posible extraer las líneas de la carretera correctamente dado que no sería sencillo diferenciarlas del resto de los elementos de la escena.

Lo que se espera es que el resultado se pueda replicar en cualquier escenario. En este caso se realizará sobre un escenario controlado para reducir la complejidad del problema.

El entorno sobre el que se va a trabajar está compuesto por una serie de elementos:

- **Circuito**: En esta fase el circuito está compuesto únicamente por dos líneas blancas que simulan las líneas de una carretera.
- **Escenario objetivo:** Es el escenario en el que se desea utilizar la red de extracción de características de las líneas y el aprendizaje de la conducción. En el caso de este proyecto, este escenario recibe el nombre de *sótano*.
- **Escenario *Chroma* o artificial**: Permite generar imágenes aumentadas. Este escenario artificial está compuesto por colores fácilmente separables. Teniendo el circuito en el interior de este escenario, es posible capturar imágenes y procesarlas para eliminar todo a excepción de las líneas de la carretera.

  Teniendo aisladas las líneas de carretera, estas se pueden superponer sobre imágenes del escenario objetivo y utilizarlas en el entrenamiento. En la Figura 4.23 se muestra un ejemplo del proceso descrito.

[4] Es una técnica audiovisual utilizada para sustituir un color de una imagen o vídeo y reemplazar el área que ocupaba ese color por otra imagen o vídeo. [46]

(a) Imagen capturada en el escenario *Chroma*

(b) Imagen capturada en *sótano* (escenario objetivo)

(c) Imagen aumentada resultante

Figura 4.23: Flujo del proceso de generación de imágenes aumentadas. Primer entorno.

*Aplicando una máscara de color sobre la imagen a se aíslan las líneas de la carretera y se superponen sobre la imagen b, teniendo como resultado la imagen aumentada mostrada en la imagen c.*

El entrenamiento del *Autoencoder* en esta fase consiste en generar las imágenes aumentadas como entrada de la red y las líneas de carretera de dichas imágenes como salida esperada de la red. A partir de ese entrenamiento no supervisado, el *Autoencoder* debería ser capaz de aislar las líneas de la carretera en el escenario objetivo directamente. En la experimentación de esta fase se mostrarán figuras y se harán referencias a vídeos que faciliten la comprensión de la tarea.

### 4.6.1. Vehículo utilizado

En esta fase se utiliza el mismo vehículo que en la fase final, por lo que será en ese capítulo donde se describirá detalladamente el vehículo utilizado.

Un aspecto importante que se modifica respecto a fases anteriores es la posición de la cámara. Esta se coloca en una posición más elevada y el punto al que enfoca se sitúa más abajo para capturar la zona de interés.

### 4.6.2. Implementación y Herramientas

Para realizar el control y obtención de datos de entrenamiento se utiliza el sistema ya descrito en la Fase III (4.5. Fase III: Sistema de control y obtención de datos de entrenamiento). Este sistema posibilita el control supervisado del vehículo a través de un volante y visualización de vídeo en tiempo real. También guarda los frames visualizados junto con el valor de dirección para formar conjuntos de datos de entrenamiento.

Al igual que en la fase de experimentación en el entorno simulado, se utiliza OpenCV para procesar las imágenes de entrenamiento (para aislar las líneas de la carretera). Este procesamiento se realiza aplicando una máscara sobre la imagen en color. La máscara está compuesta por un límite inferior y superior del color que se quiere mantener o eliminar. El espacio de color utilizado es HSV por la facilidad que ofrece para seleccionar colores.

### 4.6.3. Entorno de trabajo

En este apartado se describirán con más detalle los elementos que componen el entorno en el que se abordan las tareas de esta fase. Esto incluye el escenario *Chroma*, el circuito y el escenario objetivo.

En esta fase aparece un nuevo concepto importante a desarrollar, el escenario *Chroma*. Para que sea posible aplicar la técnica de *Chroma Key* sobre las imágenes capturadas es necesario que los materiales del escenario tengan colores fácilmente

separables. De esta forma será sencillo reemplazar esos colores por imágenes de un escenario real y generar imágenes aumentadas.

En esta fase se realiza una aproximación, se incluyen los elementos necesarios para realizar las pruebas. Será en la fase final en la que se desarrolle un escenario más elaborado.

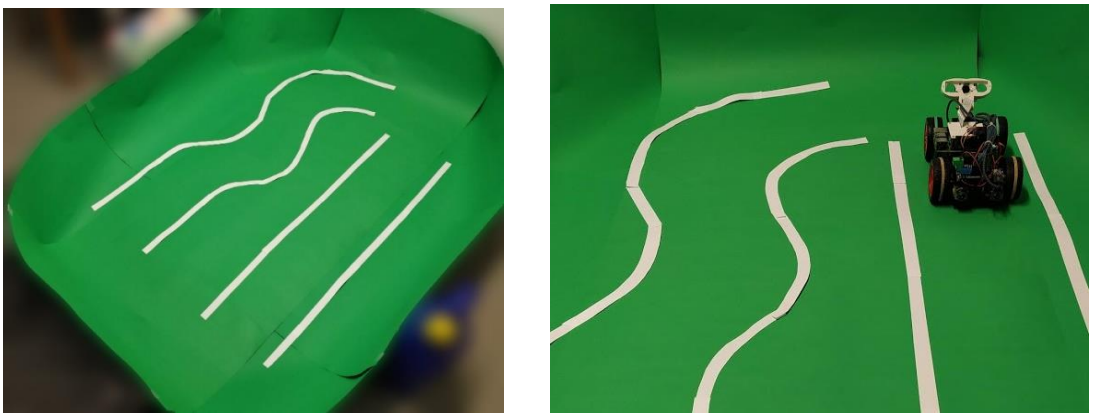
Aunque pueda parecer algo intuitivo o trivial, ha sido necesario tomar una decisión cautelosa sobre los materiales utilizados. Los factores más importantes tenidos en cuenta han sido el color y el tipo de material. También se ha atendido a las propiedades de los materiales para facilitar la construcción del escenario.

Ha sido necesario escoger entre distintos materiales, como cartulina, goma EVA, fieltro o tela. El material tiene implicaciones directas con el comportamiento de la luz, haciendo que se puedan generar más brillos, sombras, variaciones de color… etc. que causan problemas. Tras realizar algunas pruebas, las cartulinas eran el material que mejor comportamiento tenían con la iluminación. Este ha sido el material utilizado para desarrollar los escenarios.

En cuanto al color, en este escenario se utiliza únicamente uno. Al tomar esta decisión lo que se ha tenido en cuenta es que el material seleccionado tuviese el color más puro posible, de forma que la separación o eliminación de este fuese sencilla. Realizando pruebas con algunos colores como negro, rojo, azul, amarillo y verde, los que mejor resultado han mostrado han sido el verde y el rojo. Entre ellos se ha escogido el verde para desarrollar el escenario.

Existen otros muchos factores que se podrían tener en cuenta, como la iluminación, o la cámara utilizada, además de profundizar más en los ya mencionados. Sin embargo, no se ha tratado con ello en este proyecto.

El escenario *Chroma* desarrollado en esta fase se ha realizado en una especie de bañera. En su interior se encuentran dos partes de una carretera, un tramo recto y otro tramo con curvas. Se puede observar este escenario en la Figura 4.24.



(a) Escenario desarrollado en la fase IV (b) Vehículo en el interior del escenario desarrollado

Figura 4.24: Escenario *Chroma* desarrollado en la fase IV. Primer entorno.

El vehículo recorrerá el circuito en el interior del escenario *Chroma* tomando imágenes con colores fácilmente separables. Después de realizar ese paso, se tomarán imágenes en el escenario en el que se quiere entrenar la red de extracción de características (escenario objetivo). Teniendo ambos conjuntos de imágenes, se eliminará el color verde de las imágenes del escenario, quedando únicamente las líneas que serán

superpuestas sobre las imágenes del escenario objetivo, simulando una secuencia de conducción en él.

Tras entrenar la red en el escenario *Chroma* como se ha indicado anteriormente, lo interesante será comprobar si funciona en el escenario objetivo directamente. Para ello, se replican los tramos de la carretera en dicho escenario. El escenario objetivo utilizado recibe el nombre de *sótano.* Se puede observar la réplica en la Figura 4.25.

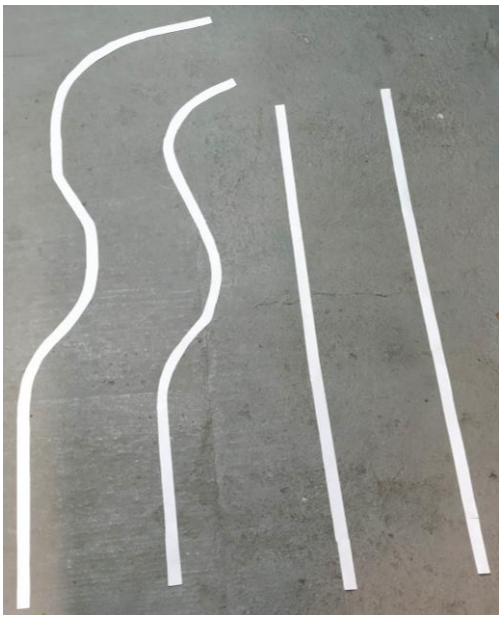


Figura 4.25: Vista desde arriba del circuito de la fase IV en el escenario objetivo, el *sótano*.

*Se realiza una réplica del circuito junto al escenario Chroma. En este caso las líneas se encuentran sobre el suelo del escenario objetivo, sin estar superpuestas de forma virtual.*

### 4.6.4. Experimentación

El primer conjunto de datos generado para realizar esta experimentación está formado por la imagen capturada en el escenario *Chroma*, líneas sobre verde, y la procesada mediante *OpenCV*, líneas aisladas. Las imágenes capturadas consisten en *frames* de secuencias de conducción en el escenario, tanto siguiendo el recorrido delimitado por las líneas como de forma arbitraria para capturar multitud de perspectivas que podrían ser útiles para realizar algunas reconstrucciones. Este conjunto de datos está formado por 3.500 imágenes de entrenamiento y 1.500 imágenes de test. Las imágenes tienen un tamaño de 320x240.

El primer experimento es el 0053multigpu-escenarioP1-CCCCFDDRCtCtCtCt-0001. En este experimento se entrena un modelo introduciendo el par de imágenes del conjunto del escenario *Chroma* para probar posteriormente la reconstrucción de las líneas, es decir, en este experimento aún no se utiliza el escenario objetivo. El modelo creado utiliza 4 capas convolucionales, un vector latente de 32 valores y ha sido entrenado durante 40 ciclos.

En este experimento se obtiene un buen resultado. Se puede observar un ejemplo en la Figura 4.26 y la secuencia completa en el **Vídeo 4.1**. El vídeo está compuesto en su totalidad por *frames* de test.

Ya que la escena está compuesta principalmente de verde, el balance de blancos de la cámara no se establece correctamente y las líneas blancas parecen algo verdosas. Sin embargo, no es un problema para realizar esta experimentación.

(a) Imagen capturada (b) Imagen reconstruida

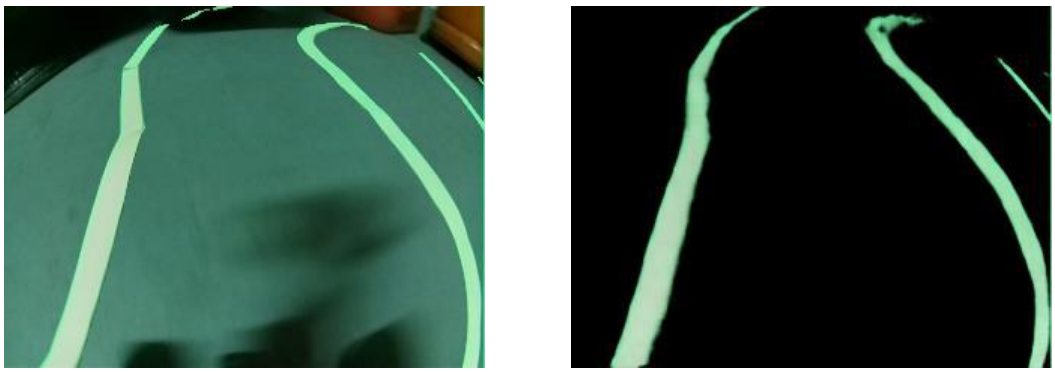
Figura 4.26: Resultados del experimento 0053.

Tras obtener resultados satisfactorios en el experimento anterior, se puede continuar con la tarea. Para ello se genera un nuevo conjunto de datos. Ahora entra en juego el aspecto más interesante de la tarea, entrenar el *Autoencoder* en el escenario objetivo generando imágenes aumentadas.

Este nuevo conjunto está formado por unas 6.500 imágenes de entrenamiento y unas 2.800 de test. Para generar las imágenes aumentadas se capturan imágenes en el *sótano*, obteniendo alrededor de 6.500. Las imágenes capturadas en el escenario *Chroma* se procesan para eliminar el color verde y obtener las líneas de la carretera. Estas se superponen sobre las imágenes del *sótano* para formar la imagen aumentada, la entrada de la red. Al igual que antes, la salida de la red es la imagen que solo contiene las líneas.

El primer experimento que hace uso de este conjunto de datos es el 0057multigpu-escenarioP4-CCCCFDDRCtCtCtCt-0001. Los parámetros de entrenamiento son los mismos que se han explicado para el experimento 0053. Se obtienen buenos resultados sobre la secuencia de test del conjunto de datos. A continuación se realizan un par de pruebas más.

La primera prueba consiste en tomar, de nuevo, una secuencia de conducción en el escenario *Chroma*, superponer las líneas extraídas sobre las imágenes de *sótano* y probar la red. Se obtienen resultados lo suficientemente buenos como para que la tarea se pueda realizar. Se pueden ver estos resultados en la Figura 4.27 y la secuencia completa en el **Vídeo 4.2**.

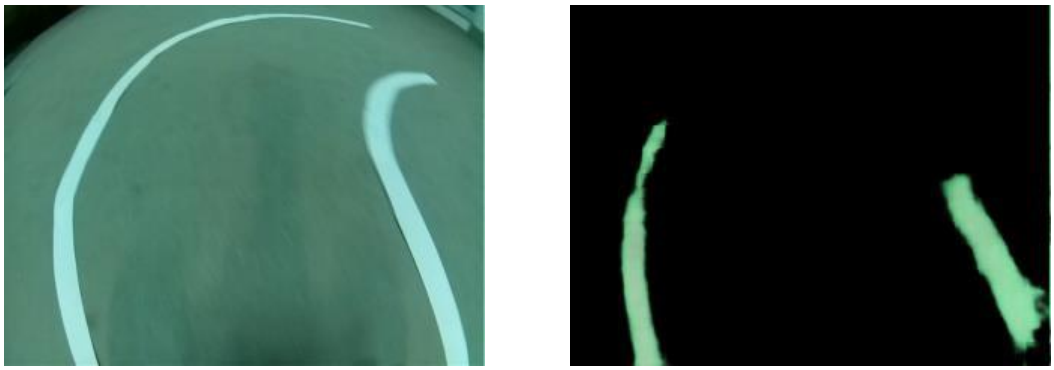


(a) Imagen aumentada (generada) (b) Imagen reconstruida

Figura 4.27: Imagen aumentada y su reconstrucción. Modelo del experimento 0057.

*En esta prueba se generan de nuevo imágenes aumentadas, es decir, se superponen las líneas de la carretera sobre imágenes del sótano. Esta prueba se realiza para probar la red con imágenes generadas diferentes a los de entrenamiento.*

La segunda prueba consiste en capturar una secuencia de conducción sobre el circuito pero en el escenario objetivo, el *sótano*. De esta forma se obtienen imágenes con las líneas de carretera reales, sin necesidad de superponer nada, está es la prueba más interesante. En este caso los resultados son mucho peores. Se pueden observar estos resultados en la Figura 4.28 y en el **Vídeo 4.3**. Estos resultados implican que la red no tiene la robustez necesaria, por lo que será necesario aplicar alguna técnica adicional.



(a) Imagen capturada en *sótano* (b) Imagen reconstruida

Figura 4.28: Prueba en sótano (escenario objetivo). Modelo del experimento 0057.

*En esta prueba se toma una secuencia de conducción sobre el circuito en el escenario objetivo, es decir, las imágenes no han sido generadas superponiendo las líneas, los elementos son reales. El objetivo es probar si la red entrenada con imágenes aumentadas funciona sobre el escenario real. No se obtienen muy buenos resultados.*

En los siguientes experimentos se realizan algunos cambios con el objetivo de mejorar los resultados obtenidos. Uno de estos cambios consiste en aumentar el tamaño del vector latente a 64 valores (experimento 0058multigpu-escenarioP4-CCCCFDDRCtCtCtCt-0002). Sin embargo este cambio no tiene ninguna mejora aparente.

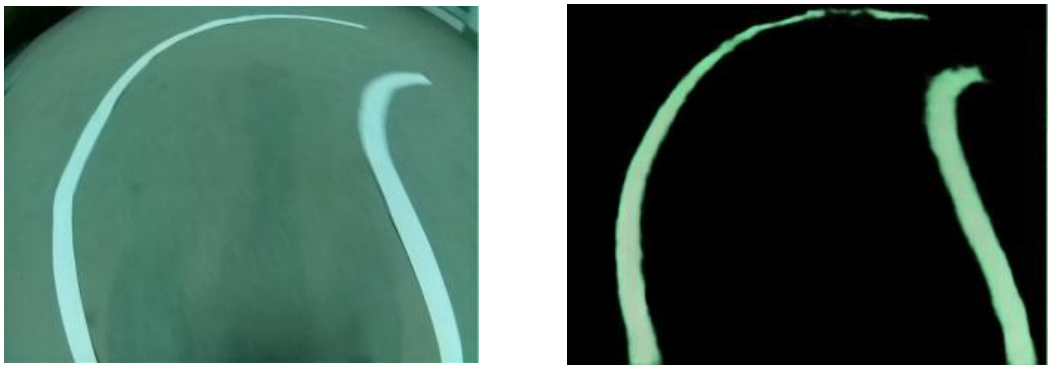
Otro de los experimentos consiste en aplicar la técnica de *data augmentation*, ensanchando y adelgazando las líneas de la carretera. Utilizar esta técnica está ligada con la robustez de una red, por lo que se cree que podría mejorar los resultados. Sin embargo, este cambio tampoco ofrece ninguna mejora.

Finalmente, se aplica otra técnica que puede mejorar la robustez de las redes de neuronas, la introducción de ruido sobre los datos de entrada. Este ruido se genera mediante una capa de *Keras* llamada *GaussianNoise*.

Inicialmente se realiza un experimento con una cantidad de ruido baja, con una desviación típica de 0,1 en la distribución gaussiana (experimento 0060multigpu-escenarioP4-CCCCFDDRCtCtCtCt-0004). Como resultado se alcanza una pequeña mejora, por lo que se continúa aumentando la cantidad de ruido introducido.

En el experimento 0061 se utiliza una desviación típica de 0,2. En este caso se observa una mejora mucho mayor de la reconstrucción en comparación con el experimento 0060. Por último, se intenta mejorar aún más el resultado. En el experimento 0062 se aumenta la desviación típica a un valor de 0,4. En este experimento se obtienen mejoras, aunque en menor medida, por lo que se establece este último como el que mejor resultados obtiene.

En la Figura 4.29 se puede observar el ejemplo anteriormente fallido pero con la reconstrucción del modelo del experimento 0062. En el **Vídeo 4.4** se pueden observar las mejoras incrementales obtenidas en los modelos 0060, 0061 y 0062.



(a) Imagen capturada (b) Imagen reconstruida

Figura 4.29: Prueba en sótano. Modelo del experimento 0062.

Adicionalmente, en el **Vídeo 4.5** se puede observar la diferencia entre el modelo que no utiliza ruido y el que sí lo hace (0057 vs 0062).

### 4.6.5. Conclusiones de la fase

En esta fase se consigue desarrollar por completo una de las principales tareas propuestas en el objetivo de este proyecto. Se ha podido comprobar como el *Autoencoder* desarrollado es capaz de aislar las líneas de la carretera en el escenario objetivo.

Uno de los aspectos que no se ha destacado han sido las diferentes técnicas o variantes de *Autoencoders* utilizados. En la red construida se han utilizado varias capas y, además, son capas convolucionales. También se ha introducido ruido en los datos de entrada. Siguiendo la teoría vista en el estado del arte, se podría decir que se está utilizando un *Denoising Stacked Convolutional Autoencoder*. Y no sólo eso, se están realizando varias tareas, principalmente una extracción de características, aunque ésta se ha mejorado realizando una segmentación o extracción de una zona de interés y eliminando ruido de la entrada (*Denoising*).

Uno de los aspectos más importantes tratados en esta fase ha sido la introducción de ruido en los datos de entrenamiento. Esta técnica ha sido totalmente necesaria para desarrollar esta tarea con resultados satisfactorios. Se ha podido comprobar empíricamente como la robustez de una red aumenta al utilizar ruido.

Con la red desarrollada se puede obtener un vector de características (vector latente del *Autoencoder*) de las líneas de la carretera. Dado que la reconstrucción se consigue realizar correctamente, se considera que los valores de este vector definen las líneas de la carretera, realizando una extracción de características válida. En la próxima y última fase, se desarrollará una red que utilice como entrada dicho vector de características para comprobar si es posible realizar la conducción autónoma de esta forma.

Como conclusión con respecto a los resultados obtenidos, se puede observar cómo la red desarrollada es suficientemente robusta. Esta es capaz de realizar un aprendizaje con imágenes aumentadas (superponiendo las líneas de carretera sobre imágenes del escenario objetivo) y utilizar el *Autoencoder* en el escenario objetivo para aislar dichas líneas.

# 5. DISEÑO Y EXPERIMENTACIÓN FINALES

En este capítulo se describirá el diseño y la experimentación realizada para abordar el objetivo principal del TFG en el entorno final. El contenido de este capítulo corresponde con la fase 5. El enfoque de dificultad incremental con el que se ha desarrollado el proyecto ha permitido resolver los problemas encontrados con mayor facilidad. Se incluye una tarea nueva con la que no se había tratado antes, la red neuronal encargada de aprender a realizar la conducción autónoma.

El objetivo de esta fase es integrar los conocimientos y desarrollos obtenidos en fases anteriores para obtener un vehículo capaz de aprender a conducir en base al procesamiento de imagen logrado.

## 5.1. Vehículo utilizado

En este apartado se describirá por completo el prototipo del vehículo final. Serán descritos los componentes principales y las decisiones más importantes. Para finalizar el apartado, se mostrarán imágenes y el esquema de conexiones del prototipo final.

El procesamiento principal del vehículo se realiza en una Raspberry Pi 3B+. En las fases previas se ha utilizado un modelo inferior (Raspberry Pi 3B) con el objetivo mantener en buen estado la mejor versión del SBC del que se dispone.

Una de las principales ventajas que aporta la versión 3B+ respecto a la 3B es un aumento en la frecuencia de su CPU, que pasa de 1,2 GHz a 1,4 GHz. Otra de las ventajas reside en la conectividad wifi, aunque en este proyecto se utilizará un adaptador USB con antena que dispone de mayor alcance.

La cámara utilizada finalmente ha sido la Pi Camera *rev 2.2* con una lente gran angular. Esta cámara captura imágenes con un campo de visión de 160º, una funcionalidad que ha sido necesaria para no perder del campo de visión zonas importantes de la carretera.

La orientación y posición de la cámara se han establecido realizando experimentación. La altura establecida ha sido la necesaria para ver el ancho de la carretera en todas las situaciones (rectas y curvas). La orientación se ha establecido intentado evitar capturar partes no significativas. Debido al gran campo de visión de la cámara si se orienta hacia el horizonte esta capturará el suelo, el horizonte e incluso el cielo. Por esto último, la cámara se ha orientado parcialmente hacia el suelo, capturando mayoritariamente la carretera y el horizonte.

Todos los componentes del vehículo se alimentan a partir de una fuente principal de ocho baterías AA Ni-Mh. En esta última fase de desarrollo ha sido necesario cambiar las baterías utilizadas debido a la degradación que han sufrido durante todo el proyecto. Se continúa utilizando baterías del tipo Níquel – Metal Hidruro, pero las nuevas baterías pasan a tener una capacidad de 1.600 mAh a 2.800 mAh.

Para obtener la tensión regulada ha sido necesario utilizar un módulo de conversión de voltajes. Este módulo está basado en el chip XL1509, se ha mostrado una imagen y las características del componente en el apartado de análisis. La entrada de este módulo debe ser de al menos 6,5 V para garantizar una salida constante de 5 V y 2 A. La salida de este módulo sirve para alimentar a la Raspberry Pi y el controlador de señales PWM.

Los motores de empuje se alimentan a través del módulo encargado de ellos. Este módulo está basado en el chip L298N. Necesita una corriente constante de al menos 2 A y ofrece un voltaje de alimentación de 5 hasta 35 V. La tensión de entrada de este módulo

llega directamente de las baterías, sin necesidad de regulación. Se muestra este componente en la Figura 5.1.

Tanto la dirección como la velocidad del vehículo se controla mediante señales PWM. Para realizar el control de estas señales se utiliza un módulo basado en el chip PCA9685 y mediante una interfaz de comunicación I2C, que se conecta a la Raspberry. Este módulo dispone de 16 canales de 12 bits para controlar señales PWM. La tensión de entrada necesaria para hacerlo funcionar es de entre 3 y 5 V. Se muestra este componente en la Figura 5.2.

Mediante el módulo PCA9685 se consigue aumentar la cantidad de señales PWM disponibles desde la Raspberry. Este módulo se alimenta a través de la salida de 5 V y 2 A del regulador de voltaje ya mencionado, utilizado en este caso únicamente para alimentar el servomotor de la dirección. La alimentación de la parte lógica (5V) la entrega la Raspberry en la entrada VCC.

Figura 5.1: L298N – Controlador de motores

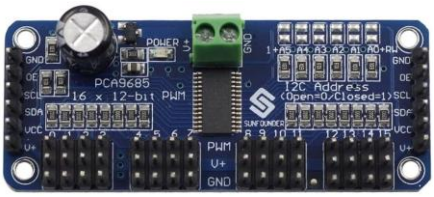



Figura 5.2: PCA9685 – Controlador PWM

El servomotor de dirección se conecta al controlador PWM que recibe la información de control de la Raspberry a través de I2C.

El controlador de los motores de empuje está conectado a 4 puertos GPIO de la Raspberry que indican el sentido hacia el que deben girar. Para establecer una velocidad de giro se utilizan dos señales del controlador PWM que se conecta al módulo de control para ajustar el ciclo de trabajo. El detalle de cómo se modifica la velocidad mediante el ciclo de trabajo se trata en el ANEXO I: SEÑALES PWM.

En cuanto a los motores de empuje utilizados, se estudiarán varias opciones en la experimentación de esta fase ya que dependen del resultado de la tarea de conducción.

En la Figura 5.3 se puede visualizar el esquema de conexiones del vehículo final.

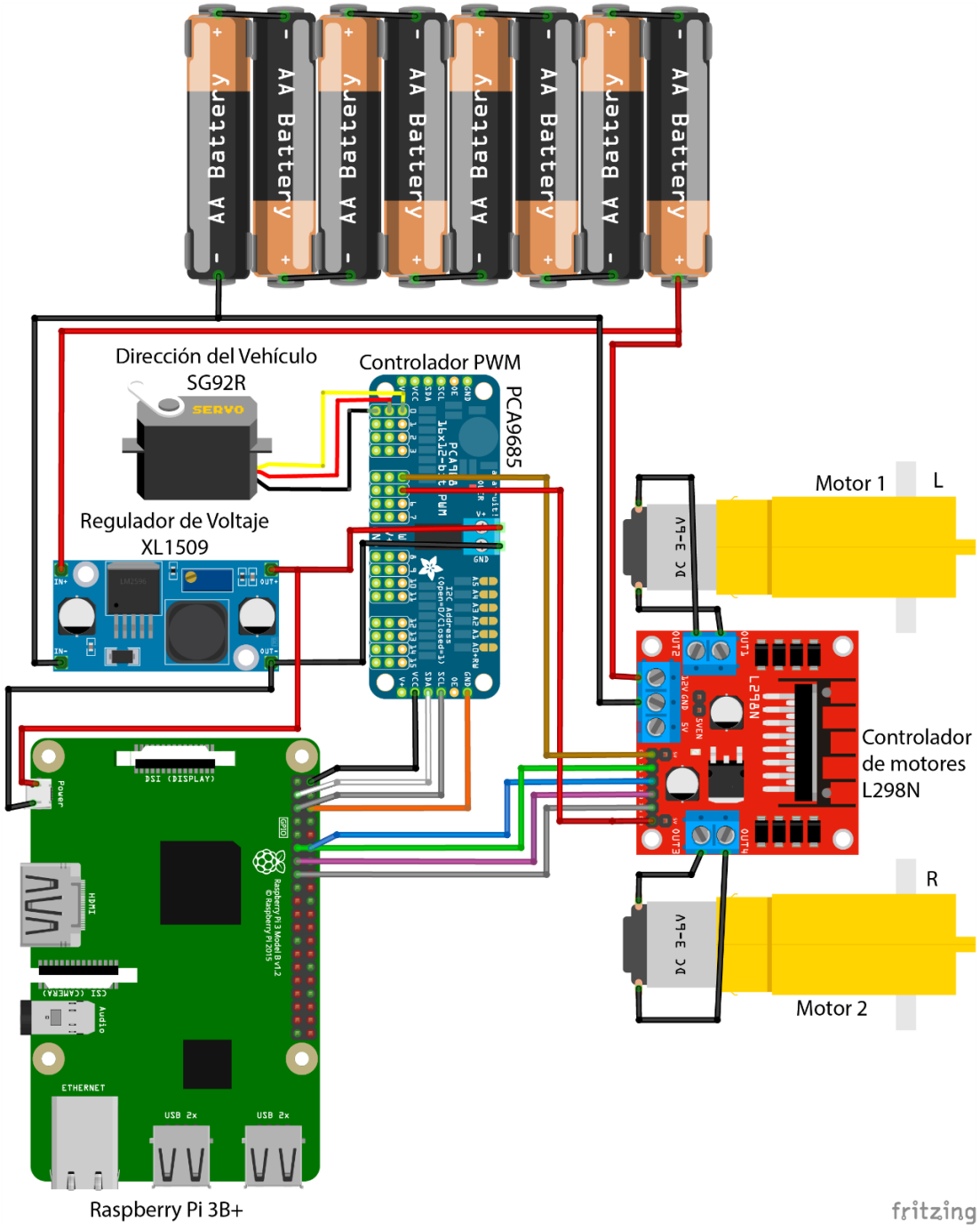
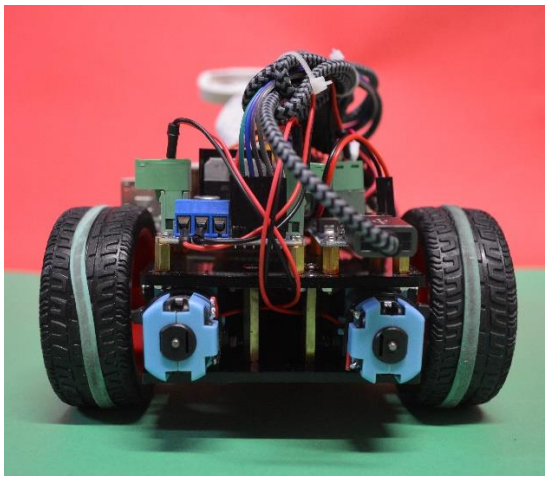
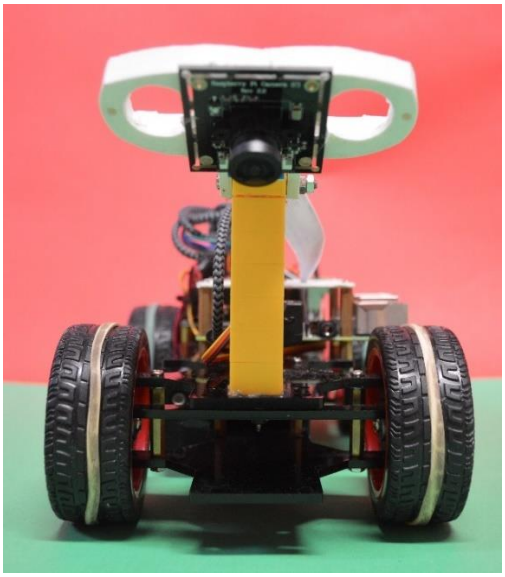



Figura 5.3: Esquema de conexiones del vehículo.

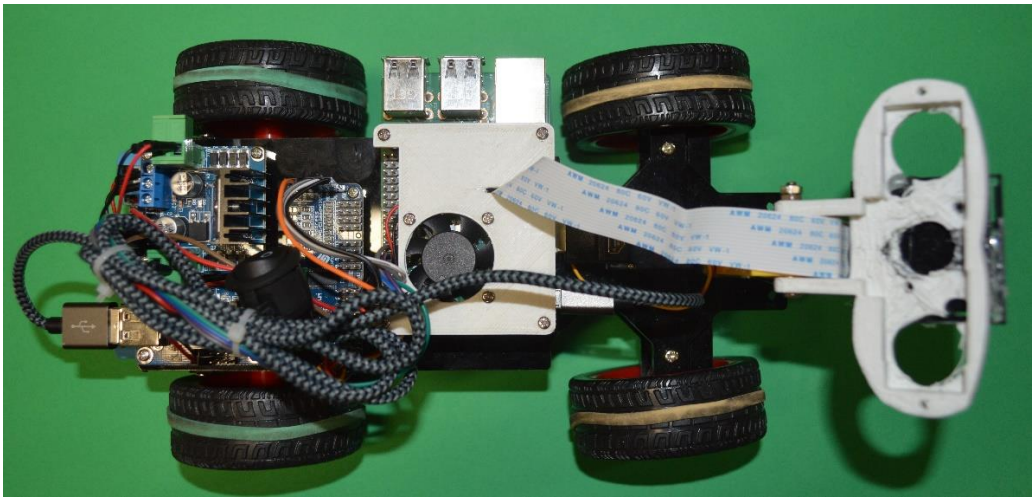
En la Figura 5.4 se pueden observar múltiples vistas del prototipo final del vehículo.



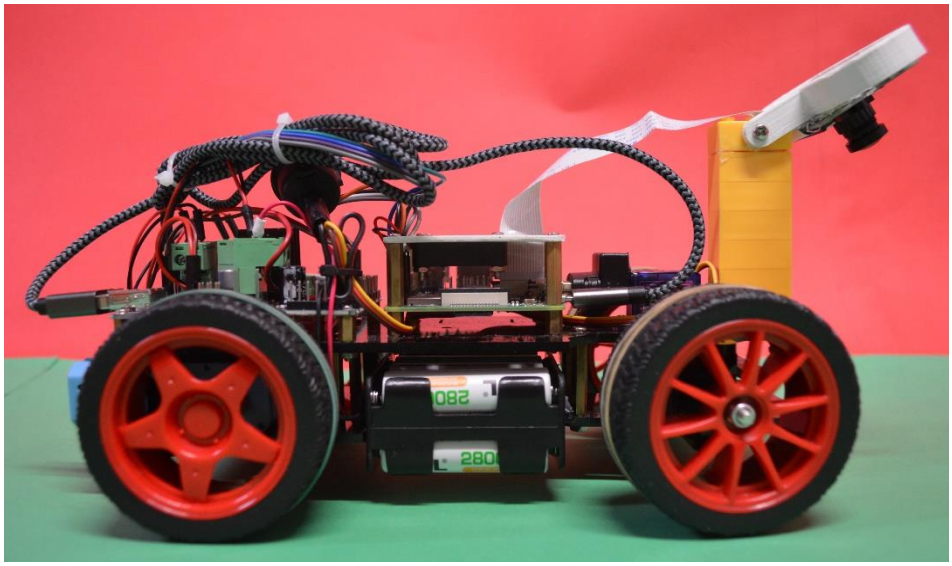
(a) Vista trasera



(b) Vista frontal



(c) Vista cenital



(d) Vista lateral

Figura 5.4: Múltiples vistas del vehículo final.

## 5.2. Entorno de trabajo

Una vez se ha comprobado en la experimentación previa que la tarea de extracción de características de las líneas es viable en el primer entorno de trabajo, se desarrolla un entorno algo más elaborado para la experimentación final.

En este apartado se describirá como es el segundo entorno de trabajo desarrollado para abordar las tareas en esta fase. Esto incluye la descripción del escenario *chroma*, el circuito y el escenario objetivo.

### 5.2.1. Escenario *Chroma* y Circuito

De nuevo, se desarrolla un escenario artificial cerrado como si fuese una bañera y con colores sólidos para aplicar la técnica de *Chroma Key*. En este caso se colocan los laterales con mayor altitud para evitar que la cámara capture elementos externos. Las medidas de este escenario son 45 cm de alto, 200 cm de largo y 170 cm de ancho.

Se utilizan dos colores diferentes, verde para el suelo y rojo para el cielo. Separar ambas partes con colores diferentes permite identificarlas con facilidad. Esta funcionalidad está pensada de cara a una tarea futura de creación de escenas virtuales realistas. Una mejora colateral de esto ha sido que la combinación de colores del escenario permite operar correctamente al balance de blancos automático de la cámara.

El circuito que se desarrolla para esta experimentación contiene las líneas de la carretera sobre cartulina negra que imita el asfalto. En este caso, el recorrido es totalmente cerrado para facilitar la tarea de entrenamiento dado que la conducción se realiza sin tener que salir de él. El circuito tiene una forma rectangular con sus vértices curvados y con una complicación añadida. Una de las rectas del circuito se cambia por una especie de chicane[5] haciendo que sea necesario realizar giros en ambas direcciones rápidamente.

Este circuito se ha desarrollado por piezas, haciendo posible que se puedan montar diferentes circuitos y que el transporte o almacenaje de este sea sencillo. En este proyecto solo se abordará el diseño del circuito mostrado en la Figura 5.5.

Figura 5.5: Escenario *Chroma* y circuito final. Segundo entorno.

[5] En un circuito de carreras de vehículos motorizados, serie de curvas pronunciadas cuyo fin es reducir la velocidad. Fuente: RAE.

### 5.2.2. Escenario objetivo - *Sótano*

La red de extracción de características de líneas de la carretera (el *Autoencoder*), al igual que en la fase anterior, será entrenada mediante imágenes aumentadas. Estas imágenes se toman en el escenario *chroma*, se eliminan los colores sólidos y se superpone el resultado (la carretera) sobre el escenario en el que se quiere entrenar, en este caso, el *sótano*.

Una vez entrenado el *Autoencoder*, será necesario que sea capaz de funcionar sobre el *sótano* directamente, es decir, sin utilizar imágenes aumentadas. Para ello, el circuito desarrollado es móvil y se puede desplazar desde el escenario *chroma* al *sótano*.

El escenario *chroma* se encuentra situado justo al lado de lo que recibe el nombre de escenario objetivo o *sótano*. La disposición de los elementos mencionados en el entorno de trabajo se muestra en la Figura 5.6.

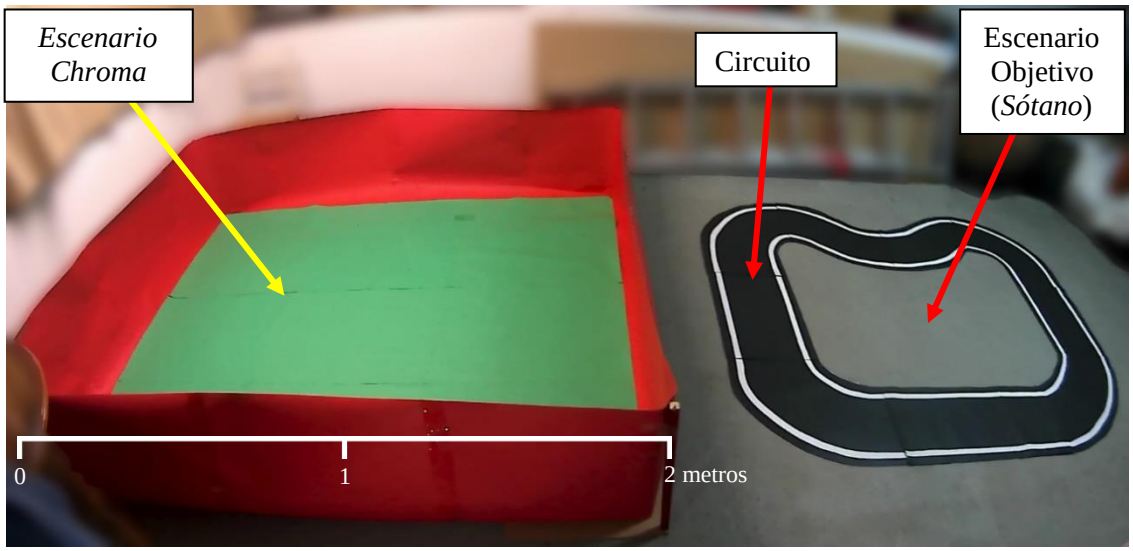



Figura 5.6: Disposición de los elementos en el segundo entorno de trabajo.

*En la imagen mostrada se puede visualizar el entorno final en el que se ha trabajado. El circuito se puede colocar en el escenario chroma o en el objetivo (sótano) en función de lo que se quiera realizar. Se incluye una línea de escala para conocer el tamaño aproximado de los elementos.*

## 5.3. Implementación

En esta fase no ha sido necesario implementar nada desde cero, aunque si ha sido necesario modificar la funcionalidad de velocidad constante del sistema de control.

Para realizar el entrenamiento de la conducción autónoma ha sido necesario mantener una velocidad constante. Esta funcionalidad ya estaba contemplada en el sistema de control, pero tenía una carencia.

La potencia aplicada sobre los motores era constante en todo momento. Sin embargo, dependiendo de los motores utilizados, se pierde velocidad al tomar curvas, quedando estancado en algunos casos. Este problema se puede resolver utilizando motores con un mayor torque o realizando un cambio en el sistema de control.

Este último consiste en aumentar la potencia establecida en los motores de tracción cuando se gira, a mayor giro, mayor potencia.

Una forma sencilla de implementar esta funcionalidad es establecer una velocidad constante y sumar un término relacionado con el valor de la dirección. El valor de giro

dado por el volante se encuentra en el rango [-1, 1] y en este caso se traduce a uno más pequeño, por ejemplo [0, 0,3].

Tomando el valor de giro traducido a ese último rango y sumándolo a la potencia final del vehículo, se consigue la funcionalidad buscada. La ecuación utilizada para implementar la funcionalidad es la (5.1).

$$potencia\_final = potencia\_cte + giro\_traducido \quad (5.1)$$

*Siendo "potencia_final" el valor de potencia establecido en los motores del vehículo. "potencia_cte" es una constante de potencia, en este caso es la necesaria para mantener una velocidad mínima en rectas. "giro_traducido" es el valor de giro traducido a un rango.*

*En el proyecto se utilizan los siguientes valores:*

*potencia_cte = 0,4 (en el rango [0, 2]), rango de giro traducido = [0, 0,3].*

Utilizando motores con una reductora mayor, el torque aumenta, con lo que no habrá pérdida excesiva de potencia en curvas. Por esta razón, lo anterior puede dejar tener importancia o necesitar un aumento de velocidad menor. En la experimentación de esta fase se estudiarán otros motores.

## 5.4. Experimentación

En esta experimentación se tratará con las dos tareas principales del proyecto.

En primer lugar, se describirá la experimentación realizada en el entorno de trabajo final para aislar las líneas de la carretera y obtener el vector de características correspondiente.

En segundo lugar, se describirá la experimentación realizada para abordar la tarea de aprendizaje de conducción autónoma.

En el **Vídeo 5.1** se muestra el proceso completo de entrenamiento seguido por las dos tareas recién mencionadas.

### 5.4.1. Tarea de extracción de características de las líneas

La tarea que se quiere realizar sigue siendo la misma que se ha explicado para el primer entorno real (fase IV). Se puede observar una representación gráfica de la red que se quiere obtener en la Figura 5.7.

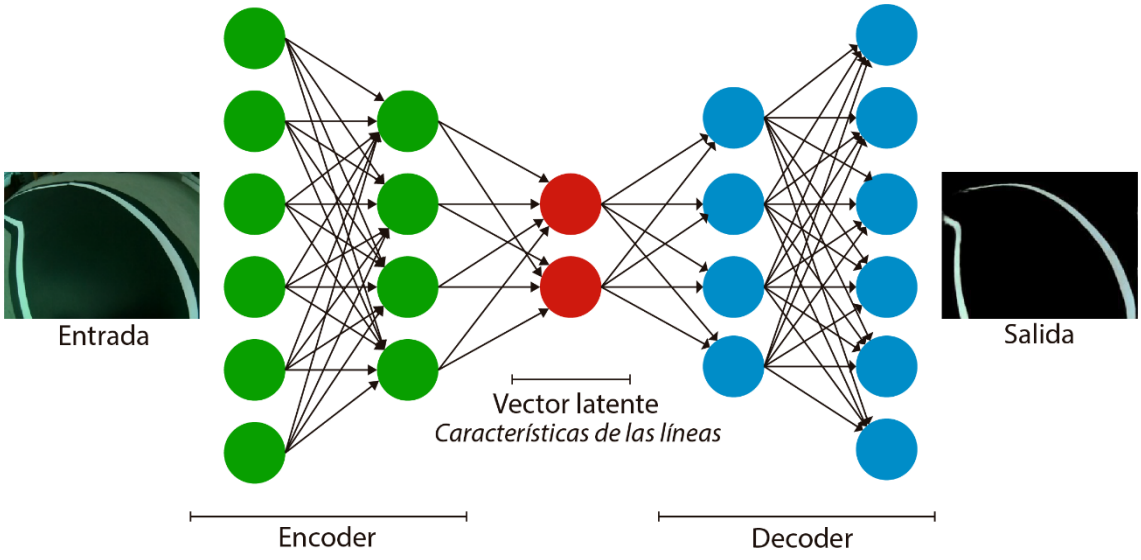
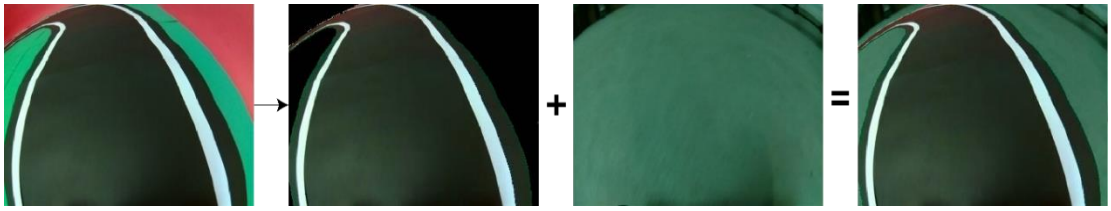



Figura 5.7: Representación gráfica de la red de extracción de líneas. (*Autoencoder*)

La tarea consiste en entrenar un *Autoencoder* que sea capaz de ignorar lo accesorio, es decir, que extraiga las líneas de la carretera. El vector latente de dicha red contendrá los valores necesarios para representar las líneas. Para entrenar la red se utilizará lo siguiente:

- Entrada: Imagen capturada en el escenario *chroma*, eliminando todo a excepción de las líneas de la carretera y el asfalto. Esta tarea se simplifica y es posible por el uso de la técnica de *Chroma Key*. El resultado se superpone sobre imágenes del escenario objetivo, el *sótano*, obteniendo imágenes aumentadas.
- Salida esperada: Imagen capturada en el escenario *chroma*, eliminando todo excepto las líneas.



(a) Imagen capturada en escenario *chroma* | (b) Imagen procesada. Carretera aislada. | (c) Imagen capturada en escenario objetivo (conducción) | (d) Imagen aumentada resultante

Figura 5.8: Flujo del proceso de generación de imágenes aumentadas. Segundo entorno.

*La imagen a ha sido capturada en el escenario chroma. La imagen c ha sido capturada en el escenario objetivo, llamado sótano. Aplicando máscaras de color sobre la imagen a se aísla la carretera (imagen b) y se superpone sobre la imagen c, teniendo la imagen aumentada resultante mostrada en la imagen d.*

Al entrenar la red con este esquema, es de esperar que sea posible tomar imágenes del circuito en el escenario objetivo (sin superponer nada, todo real) y extraer las líneas de la carretera.

Dado que ya se había realizado mucha experimentación en el primer entorno de trabajo (fase IV), el desarrollo de esta tarea en esta fase ha sido más sencillo.

El conjunto de imágenes tomadas en el escenario *chroma* (véase Figura 5.9, Imagen a) consta de unas 7.000 imágenes de entrenamiento y unas 3.000 de test, con un tamaño de 320x240. Estas imágenes pertenecen a secuencias de conducción sobre el circuito desarrollado dentro del escenario de colores. Las imágenes del conjunto se han obtenido de dos formas:

- Conduciendo correctamente por el circuito en ambos sentidos. La mayoría de las imágenes son de este tipo.
- Se incluyen secuencias de conducción arbitraría sobre el circuito. De esta forma se incluyen más perspectivas del circuito, que permitirá a la red extraer las líneas de la carretera desde más puntos de vista.

El conjunto de imágenes tomadas en el escenario objetivo (véase Figura 5.9, Imagen b) consta de unas 4.500 imágenes. Como hay mayor cantidad de imágenes de conducción en el escenario *chroma*, se repetirán algunas del conjunto del *sótano*.

(a) Imagen capturada en el escenario *Chroma*

(b) Imagen tomada en el sótano (escenario objetivo)

Figura 5.9: Ejemplos del conjunto de datos capturados en el segundo entorno de trabajo.

El primer experimento de esta fase es el 0063multigpu-escenarioVerde2P1-CCCCFDDRCtCtCtCt-0001. A diferencia de otros casos, este primer experimento no es una prueba, sino que implementa directamente la tarea que se quiere abordar.

La estructura de la red utilizada es igual a la utilizada en los últimos experimentos de la fase anterior. Esta se puede observar en el nombre completo del experimento, pero se describen algunos detalles a continuación:

- Vector latente de 32 valores.
- Cuatro capas convolucionales, cada una de ellas seguida de una capa de MaxPooling que reduce la dimensión de los datos.
- Las capas convolucionales utilizan 32, 64, 128 y 256 filtros respectivamente.
- Utiliza la técnica de ruido en los datos de entrada ya que, como se ha visto en la experimentación previa, es algo totalmente necesario. Este aspecto ha sido uno de los más importantes en este proyecto. Para la distribución Gaussiana del ruido se establece una desviación típica de 0,4.
- El entrenamiento dura 40 ciclos. El *Batch Size* utilizado para entrenar es de 32, algo que ha sido posible gracias al uso de las dos tarjetas gráficas ya mencionadas.

Un aspecto que no se ha mostrado en la experimentación previa son gráficas de descenso del error. Al entrenar este tipo de redes se ha utilizado la función de error de *binary crossentropy*. La función compara la salida esperada con la salida obtenida mediante una función que minimiza su resultado cuanto más se parecen ambas salidas. El objetivo es minimizar el resultado de esta función mientras se entrena la red.

En la Figura 5.10 se puede observar como el error ha descendido rápidamente al comenzar el entrenamiento y cómo después desciende poco a poco. El error podría descender algo más entrenando durante más ciclos, pero no en cantidades significativas.

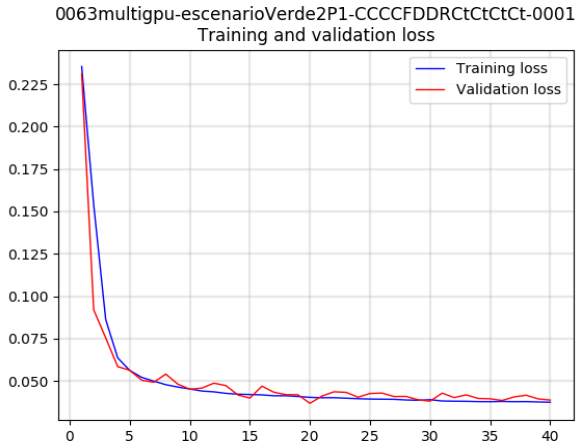
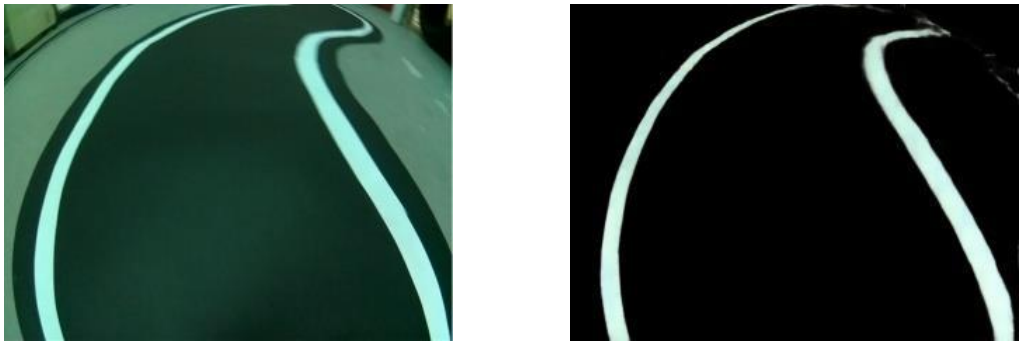



Figura 5.10: Descenso de la función de error *binary crossentropy* del experimento 0063.

Los resultados obtenidos con la red generada en el experimento anterior son bastante buenos (véase Figura 5.11), aunque se ha tenido un problema con el escenario *chroma* utilizado que se comentará a continuación.



(a) Imagen tomada en *sótano* (b) Imagen reconstruida

Figura 5.11: Resultados del experimento 0063. Entrada del escenario sótano.

*La imagen (a) es real, capturada sobre el circuito en el escenario objetivo, sótano. La imagen (b) es la reconstrucción del* Autoencoder.

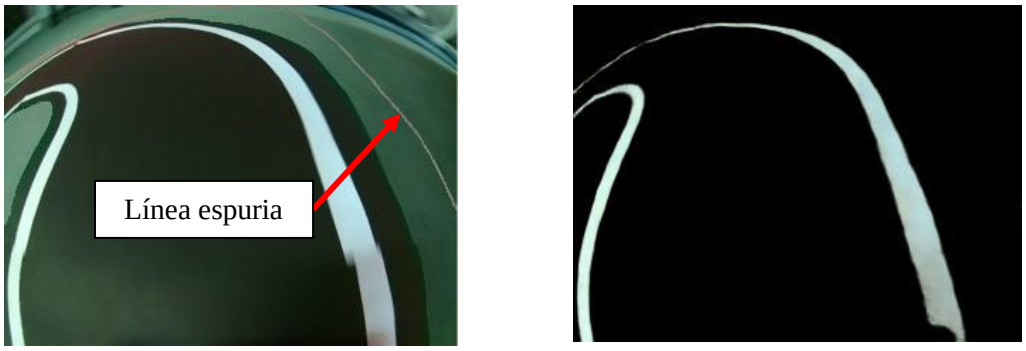
Utilizar dos colores para el escenario *chroma* ha generado un problema al procesar las imágenes para aislar la carretera. El punto en el que el color verde y rojo se unen genera en las imágenes una pequeña línea grisácea-blanquecina. El problema está en que al ser de ese color, una máscara también eliminaría parte del asfalto (gris) o de las líneas de la carretera (blanco). Se puede observar esta línea espuria en la Figura 5.12, Imagen a.

Este problema se ha achacado a la falta de nitidez de las imágenes o a algún otro aspecto de la cámara. Como la línea generada es muy fina, se ha intentado eliminar aplicando filtros de desenfoque para eliminarla, aunque sin mucho éxito.

Finalmente, no ha sido necesario resolver el problema ya que prácticamente no influye sobre los resultados finales. Lo siguiente podría ser una posible explicación a este suceso.

La línea generada es de un color grisáceo. La máscara utilizada para las imágenes que mantienen el asfalto (entrada de la red) mantiene también la línea espuria. La máscara utilizada para las imágenes que sólo mantienen las líneas de la carretera (salida esperada) sí que permite eliminar la mayor parte de dicha línea. Por lo que la red aprende a

eliminarla, sin repercutir sobre el resultado obtenido. En la Figura 5.12 (a), se puede observar la línea espuria.




(a) Imagen aumentada (generada) (b) Imagen reconstruida

Figura 5.12: Resultados del experimento 0063. Imágenes aumentadas. Línea espuria.

*Se captura la carretera en el escenario chroma, se procesa, y se superpone sobre una imagen del escenario objetivo, así se obtienen las imágenes aumentadas como la mostrada en la imagen (a). La imagen (b) es la reconstrucción alcanzada por la red. En esta figura se muestra la línea espuria de la que se ha hablado y como no repercute en el resultado.*

En el índice de vídeos de la Fase V se pueden visualizar dos vídeos que ofrecen más detalle sobre la experimentación realizada.

En el **Vídeo 5.2** se puede observar una secuencia de test con imágenes aumentadas o generadas (tomadas en el escenario *chroma*, procesadas y superpuestas en imágenes de *sótano*) y su reconstrucción.

En el **Vídeo 5.3** se puede observar una secuencia de test con imágenes tomadas en el escenario objetivo (totalmente reales) y su reconstrucción.

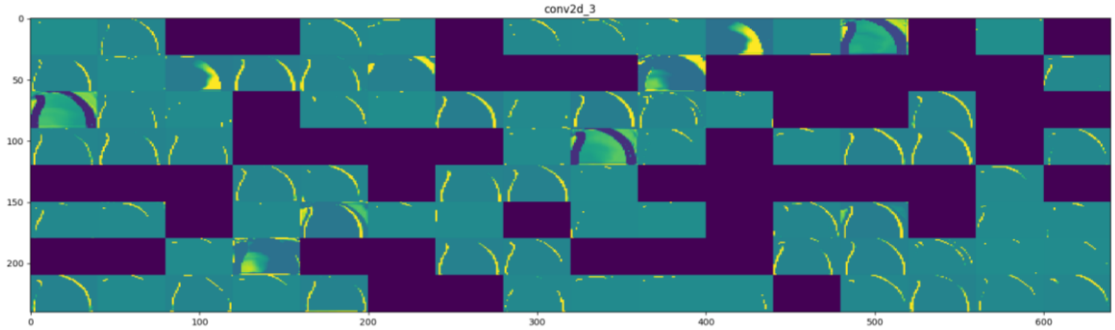
Uno de los cambios probados para intentar mejorar los resultados ha sido aumentar el ruido introducido a los datos de entrada. Este cambio se ha probado en el experimento 0064multigpu-escenarioVerde2P1-CCCCFDDRCtCtCtCt-0002. En este se introduce ruido con una desviación típica de 0,5 en la distribución. Al no obtener ninguna mejora, se descarta la opción.

Aprovechando el conocimiento adquirido en la experimentación previa con *Deep Learning* sobre técnicas de visualización de redes de neuronas, se ha aplicado una ellas en este dominio. (Estas técnicas han sido documentadas en el ANEXO III: EXPERIMENTACIÓN PREVIA, C.1.1. Visualización de redes de neuronas)

Se ha utilizado la técnica de visualización de activaciones intermedias. La técnica de GRAD-CAM para obtener mapas de calor no se ha podido aplicar en este caso ya que solo puede utilizarse sobre redes que clasifican por clase.

En la Figura 5.13 se puede observar la activación de los filtros o canales de la tercera capa convolucional del *Autoencoder*. Estos se han obtenido a partir de una imagen de conducción entrando en un giro hacia la izquierda. Se puede observar cómo cada filtro actúa sobre la imagen de entrada aislando características que, en este caso, son principalmente de las líneas.

(a) Imagen de entrada del *Autoencoder*




(b) Filtros de la tercera capa convolucional. Extracción de líneas de carretera

Figura 5.13: Visualización de activaciones intermedias en el *Autoencoder*.

### 5.4.2. Tarea de aprendizaje de conducción autónoma

En este apartado se describirá la experimentación realizada sobre la tarea de aprendizaje de conducción autónoma.

Consiste en el control autónomo de la dirección del vehículo. La red de neuronas que aparece en esta tarea no utiliza como entrada una imagen, sino que utiliza las características extraídas por el *Autoencoder*. El aprendizaje de conducción se realiza una vez que se dispone del AE entrenado, con lo que se realiza en el escenario objetivo (*sótano*).

Para realizar el entrenamiento de esta red se parte de secuencias de conducción en las que se almacena la imagen capturada y el valor del eje de dirección en ese momento.

La red será entrenada con las características de las líneas de la carretera como entrada y el valor de dirección como salida esperada. De esta forma, se podrá saber si la red es capaz de aprender a inferir el giro a realizar a partir de las características de las líneas.

Para construir esta red se aprovecha la parte del *Autoencoder* que extrae las características. La parte aprovechada es el *Encoder*, que obtiene el vector latente a partir de la imagen capturada. Se utiliza el vector (las características) como la entrada de la red densamente conectada, la red de decisión. Al realizar el entrenamiento, los pesos del *Encoder* se "congelan" para que no varíen. Se puede observar la estructura de la red completa en la Figura 5.14.

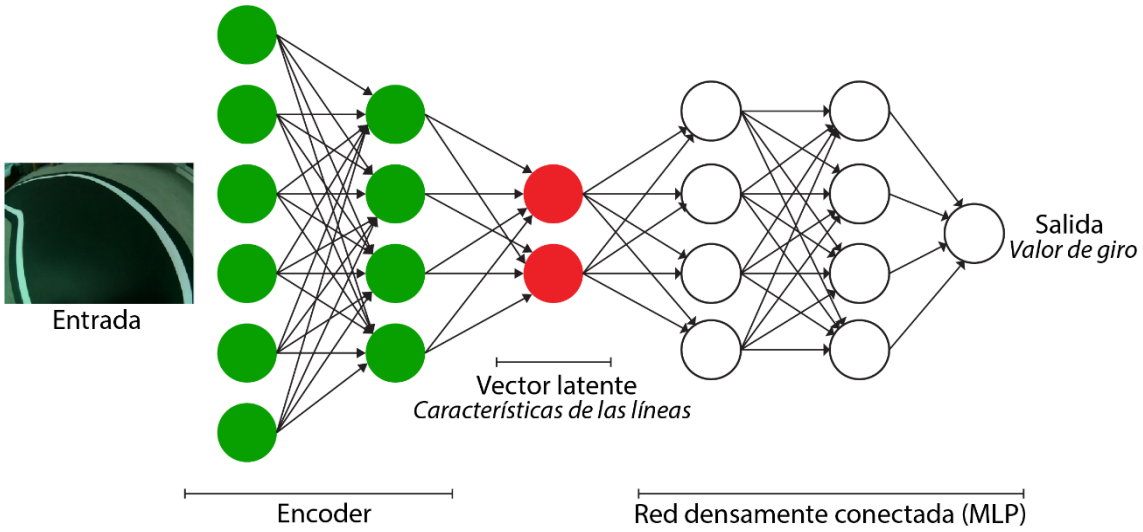



Figura 5.14: Representación gráfica de la red de conducción completa.

El primer conjunto de datos se obtiene siguiendo el recorrido del circuito en ambos sentidos. Se capturan un total de 18.700 *frames* de conducción, resultado de conducir unos 6 minutos en cada sentido. Este conjunto se divide en dos partes, 70% para entrenar y 30% para realizar el test. El tamaño de las imágenes es el que utiliza el *Autoencoder*, 320x240.

El primer experimento con el que se aborda esta tarea es el 0065-carreteraSotano1-DDrDDrDDrDDrD-0001. Esta red tiene una estructura de 5 capas densamente conectadas que se alternan con capas de Dropout. Las capas densamente conectadas tienen 512, 256, 128, 64 y 1 neuronas respectivamente. Cada capa de Dropout tiene un valor de probabilidad de 0,2. Todas las capas utilizan la función de activación ReLU a excepción de la última, que indica la salida por lo que tiene una función de activación lineal. El entrenamiento tiene una duración de 20 ciclos.

En este caso la métrica utilizada es el error dado por la función MSE (*Mean Squared Error*). En la Figura 5.15 se puede observar la gráfica de disminución del error para el experimento 0065. En ella se puede ver como el error disminuye rápidamente. Esta es una de las ventajas de haber dividido el trabajo en dos tareas (extracción de características y aprendizaje de la conducción), la red de decisión tiene una entrada pequeña, con menos pesos, por lo que el entrenamiento es más sencillo y rápido.

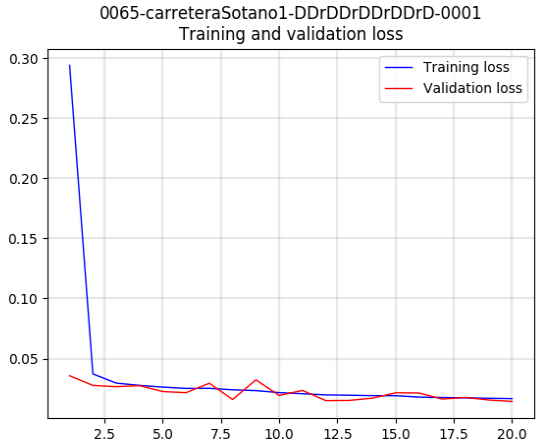



Figura 5.15: Grafica de error. Experimento 0065.

Una vez entrenada la red mencionada, se implementa el agente en el vehículo para probar el funcionamiento. Se establece una velocidad mínima y constante y se inicia el agente de conducción. Los resultados de este experimento no son muy satisfactorios. El vehículo se sale del recorrido en la parte más compleja, la chicane.

Las causas de este problema se atribuyen al tiempo de inferencia y a la velocidad del vehículo. El SBC consigue inferir unas 2,5 imágenes por segundo, siendo una cantidad muy pequeña. La velocidad del vehículo es la mínima permitida por los motores, aunque si fuese inferior, el vehículo podría no tener problemas en la conducción.

Para solucionarlo se proponen dos opciones. La primera de ellas consiste en estudiar otros motores con un mayor torque que permitan que el vehículo lleve una velocidad inferior. La segunda opción consiste en mejorar el tiempo de inferencia. Aquí se abordan ambas opciones, empezando por la segunda.

Para reducir el tiempo de inferencia se propone reducir a la mitad el tamaño de la imagen de entrada, del actual 320x240 a 160x120. Para ello, se vuelve a entrenar el *Autoencoder* con esta modificación. Este cambio se realiza sobre el experimento 0066multigpu-escenarioVerde2P2-CCCCFDDRCtCtCtCt-0001. Los resultados de éste son correctos, por lo que la reducción del tamaño de entrada se puede considerar útil.

Con este nuevo *Autoencoder* se aborda de nuevo la tarea de conducción. Este experimento es el 0067-carreteraSotano1-DDrDDrDDrDDrD-0002, que tiene las mismas características que el anterior. En este caso, se consigue un agente capaz de actualizar la dirección del vehículo unas 5 veces por segundo, aproximadamente el doble que antes. Aunque sea poco, podría ser suficiente.

Al probar el agente creado en el experimento 0067, el vehículo mejora en algunas zonas del circuito, pero sigue saliéndose del recorrido en algunas ocasiones. En este caso se cree que el problema podría estar relacionado con el entrenamiento realizado.

En la fase de conducción supervisada no se había realizado un trazado de curvas del todo correcto, lo cual tiene algunas consecuencias esenciales. Para mejorar este aspecto, por ejemplo, tomando una curva hacia la derecha por el exterior (en este caso, por la izquierda), se conseguiría tener un campo de visión más amplio de la curva y una salida más limpia, facilitando la conducción.

Con esta modificación se obtiene un nuevo conjunto de datos en el que se divide la conducción en el sentido horario y antihorario para abordar el problema de forma incremental. En primer lugar, se abordará el sentido horario.

El conjunto de datos en sentido horario tiene una cantidad de unas 11.400 imágenes. Se divide en un 70 % para entrenamiento y un 30 % para realizar test. El tamaño de las imágenes sigue siendo 160x120.

El primer experimento en este nuevo conjunto es el 0068-carreteraSotano2H-DDrDDrDDrDDrD-0001. Los resultados de este son satisfactorios, consiguiendo recorrer el circuito completo en sentido horario sin salirse del recorrido, como se puede visualizar en el **Vídeo 5.4**. Además, se puede observar como el agente está imitando el comportamiento (*Behavioral Cloning*), realizando un trazado de curvas con la técnica modificada.

El conjunto de datos en sentido antihorario se utiliza en el experimento 0071-carreteraSotano2A-DDrDDrDDrDDrD-0001. Los resultados no son tan satisfactorios como el anterior. En este caso, el agente consigue recorrer el circuito, pero con algunas

dificultades en la zona de la chicane, en la que en algunos casos se sale del recorrido. El agente consigue recorrer el resto del circuito adecuadamente.

La zona del circuito de la chicane es la más complicada, ya que requiere varios giros consecutivos y uno de ellos es muy cerrado. Por esta razón, el entrenamiento de esta zona puede haber sido más complicado para el agente. Esto se ha podido observar tanto en este experimento como en los anteriores. Además, el inicio de este tramo en el sentido antihorario es algo más cerrado, siendo más complicado este sentido del recorrido.

En la parte final del **Vídeo 5.4** se puede observar una secuencia de conducción autónoma en el sentido antihorario. En esta secuencia el agente consigue completar el recorrido sin salirse.

Por último, se unifican los dos conjuntos anteriores para determinar si se puede obtener un agente que sea capaz de recorrer el circuito correctamente en ambos sentidos. Esto se realiza en el experimento 0072-carreteraSotano2HA-DDrDDrDDrDDrD-0001. Los resultados de este agente son los mismos logrados anteriormente. En el sentido horario recorre el circuito sin problemas, y en el sentido antihorario no siempre se consigue superar la chicane.

Se ha estudiado utilizar otros motores. Hasta ahora se han utilizado los TT Motor DC 1:48. En la Tabla 5.1 se muestran las características de las opciones propuestas.

Tabla 5.1: Características principales de los motores propuestos.

| | **Características principales** | | | |
|---|---|---|---|---|
| **Motor** | **Rango de tensión** | **Reductora** | **Velocidad máxima teórica (en vacío)*** | **Velocidad máxima (con carga)*** |
| TT Motor DC 1:48 | 3 – 6 V | 1:48 | 250 RPM | 230 RPM |
| TT Motor DC 1:90 | 3 – 6 V | 1:90 | 120 RPM | 85 RPM |
| TT Motor DC 1:120 | 3 – 6 V | 1:120 | 80 RPM | 50 RPM |

***Nota.** La velocidad máxima se ve afectada por el mayor rozamiento de los engranajes al aumentar la reductora y por la carga del vehículo.*

(a) TT Motor DC 1:48

(b) TT Motor DC 1:90

(c) TT Motor DC 1:120

Figura 5.16: Motores de tracción analizados.

La diferencia principal entre los motores que se han estudiado es la reductora que utilizan. Estos tienen la misma estructura y dimensiones, es decir, se pueden utilizar unos u otros en el vehículo sin problemas de montaje.

La reductora del motor influye directamente en el torque que puede desarrollar el vehículo. Cuanto mayor sea la reductora (1:48 < 1:120), mayor será el torque, pero menor será la velocidad de avance.

- **TT Motor 1:48:** Para que el vehículo pueda realizar la conducción autónoma ha sido necesario establecer la velocidad mínima, unos 80 RPM. Incluso con esta velocidad y realizando inferencia en el vehículo, el agente no ha sido capaz de conducir adecuadamente en todos los casos.
- **TT Motor 1:120:** Estos motores tienen una reductora mucho mayor, lo que provoca que apenas pierda potencia al tomar curvas, pero resulta demasiado lento. La velocidad máxima de estos motores es de unos 50 RPM. El vehículo utiliza estos motores en el **Vídeo 5.4**.
- **TT Motor 1:90:** La velocidad máxima del vehículo con estos motores es de unos 85 RPM y la velocidad mínima de unos 30 RPM. Como se puede ver, queda aproximadamente entre la que ofrecen los dos anteriores.

Se seleccionan los motores con la reductora de 1:90 ya que el rango de velocidad que ofrece es bastante interesante para este caso. En concreto, el agente desarrollado consigue conducir adecuadamente desde la velocidad mínima, 30 RPM (0,1 m/s, en escala real equivale a unos 6 km/h), hasta unos 65 RPM (0,22 m/s, en escala real equivale a unos 12,5 km/h). En caso de aumentar la velocidad, el agente no tiene tiempo suficiente para inferir y comienza a salirse del recorrido.

En el **Vídeo 5.5** se muestra el comportamiento del vehículo equipado con estos motores. La velocidad del vehículo en el vídeo es la máxima posible realizando la inferencia en el SBC equipado, mencionada anteriormente.

## 5.5. Conclusiones de la fase

En esta última fase se ha podido unificar todo el trabajo realizado durante el proyecto para alcanzar el objetivo marcado.

La tarea de extracción de características de las líneas de la carretera se ha resuelto con mayor agilidad en esta fase. Esto ha sido posible gracias a la experimentación previa realizada.

Una vez completada esta parte del sistema de conducción autónoma, ha sido posible abordar la tarea en la que se realiza la conducción. Se creía que esta tarea sería más sencilla que la anterior pero se ha podido observar que no resultaba ser así. Aun así, se han conseguido buenos resultados.

Al utilizar un SBC pequeño ha aparecido un problema que se podía intuir, el tiempo de inferencia era superior a lo necesario. Reduciendo el tamaño de la imagen se ha podido reducir este tiempo y obtener mejores resultados.

Otro de los problemas ha sido el entrenamiento. Al estar ante una tarea de aprendizaje supervisado ha sido muy importante realizar el entrenamiento con un estilo de conducción adecuado. Realizando un trazado de curvas que permite una conducción más suave y limpia ha sido posible obtener un agente capaz de recorrer el circuito adecuadamente.

Finalmente se ha tratado con los motores, aspecto que se había dejado para el final. Era necesario conocer el tiempo de inferencia del agente para utilizar unos motores con un rango de velocidad adecuado. Se han escogido unos motores que ofrecen un rango de velocidad en el que el agente puede conducir adecuadamente, sin que surjan problemas por falta de torque.

En el capítulo de validación se tratará con casos especiales mediante los que se podrá conocer el alcance del sistema desarrollado.

# 6. VALIDACIÓN

En este apartado se presenta el plan de pruebas elaborado para validar los requisitos enunciados en el análisis del problema. Las pruebas buscan confirmar que se ha tratado con todos los requisitos y evaluar hasta qué punto se ha cumplido con el objetivo del proyecto.

La mayoría de las pruebas se han realizado al final, pero algunas otras se han realizado durante el proyecto. Las realizadas durante el proyecto han comprobado funcionalidades o tareas necesarias para la continuidad del trabajo.

En el repositorio del proyecto [39] se ha incluido un apartado con imágenes y vídeos del resultado de algunas de las pruebas realizadas en este capítulo.

## 6.1. Plan de pruebas

Las pruebas elaboradas se han clasificado en los tres grupos mencionados en la especificación de requisitos:

- **Entorno de trabajo (E)**: Hace referencia a aspectos del escenario *chroma*, el escenario objetivo y el circuito utilizado para entrenar las redes de neuronas.
- **Vehículo / Sistema de control (V/SC):** Hace referencia al vehículo y al sistema de control.
- **Sistema de conducción autónoma (SCA):** Hace referencia a todo lo relacionado con redes de neuronas y el agente desarrollado.

Cada prueba quedará definida en una tabla con los siguientes campos:

- **Identificador**: Nombre que identifica unívocamente a la prueba. Sigue una nomenclatura específica.
    - **P-*Grupo*-XX** (Prueba del *Grupo* número XX). *Grupo*: E, V/SC o SCA.
- **Objetivo**: Nombre o breve descripción de lo que se quiere comprobar en la prueba.
- **Requisito(s)**: Requisito o requisitos que trata de validar la prueba.
- **Procedimiento**: Descripción de los pasos a seguir para realizar la prueba.
- **Resultado esperado**: Describe el resultado que se debería obtener.
- **Resultado obtenido**: Indica si se ha obtenido el resultado esperado. En caso contrario, describe el resultado obtenido.

En la Tabla 6.1 se puede observar la plantilla que será utilizada para la definición de cada prueba.

Tabla 6.1: Plantilla de definición de pruebas.

| **P-*Grupo*-XX** | |
|---|---|
| **Objetivo** | |
| **Requisito(s)** | |
| **Procedimiento** | |
| **Resultado esperado** | |
| **Resultado obtenido** | |

### 6.1.1. Pruebas referentes al entorno de trabajo

En este apartado se describirá el plan de pruebas elaborado para validar los requisitos relacionados con aspectos del escenario *chroma*, el escenario objetivo y el circuito utilizado para entrenar las redes de neuronas.

Tabla 6.2: P-E-01

| **P-E-01** | |
|---|---|
| **Objetivo** | Comprobar si el escenario *chroma* del segundo entorno de trabajo (el final) permite eliminar mediante OpenCV todos los elementos a excepción de las líneas de la carretera. |
| **Requisito(s)** | RNF-12 |
| **Procedimiento** | 1. Se obtiene una secuencia de conducción sobre el circuito en el escenario *chroma*.<br>2. Se fijan las máscaras de color necesarias para eliminar todos los elementos a excepción de las líneas.<br>3. Se procesan todas las imágenes del conjunto obtenido. |
| **Resultado esperado** | Las imágenes resultantes deben disponer únicamente de las líneas de la carretera. |
| **Resultado obtenido** | Se obtiene el resultado esperado. |

Tabla 6.3: P-E-02

| **P-E-02** | |
|---|---|
| **Objetivo** | Comprobar si el escenario *chroma* del segundo entorno de trabajo (el final) permite eliminar todos los elementos a excepción de las líneas de la carretera y el asfalto. |
| **Requisito(s)** | RNF-12 |
| **Procedimiento** | 1. Se obtiene una secuencia de conducción sobre el circuito en el escenario *chroma*.<br>2. Se fijan las máscaras de color necesarias para eliminar todos los elementos a excepción de las líneas de la carretera y el asfalto.<br>3. Se procesan todas las imágenes del conjunto obtenido. |
| **Resultado esperado** | Las imágenes resultantes deben disponer únicamente de las líneas de la carretera y el asfalto. |
| **Resultado obtenido** | A excepción del algunos elementos no deseados en la imagen, se obtiene el resultado esperado.<br>Los elementos no deseados no afectan al proceso de aprendizaje, por lo que no se tienen en cuenta. |

### 6.1.2. Pruebas referentes al vehículo y el sistema de control

En este apartado se describirá el plan de pruebas elaborado para validar los requisitos relacionados con el vehículo y el sistema de control.

Tabla 6.4: P-V/SC-01

| **P-V/SC-01** | |
|---|---|
| **Objetivo** | 1. Comprobar que el sistema de control permite controlar el vehículo de forma remota. |

| | |
|---|---|
| | 2. Comprobar los valores de control ofrecidos por el HUD del vídeo en tiempo real.<br>3. Comprobar que el vídeo capturado permite visualizar las líneas que limitan la carretera en cualquier situación dentro del recorrido. |
| **Requisito(s)** | RF-02, RF-05 y RNF-06. |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Utilizando el vídeo en tiempo real, el operario conduce el vehículo ejecutando diferentes maniobras:<br>a. Movimiento hacia delante.<br>b. Movimiento hacia atrás.<br>c. Giros variables hacia ambos lados.<br>d. Combinaciones de los anteriores |
| **Resultado esperado** | 1. El operario debe ser capaz de controlar el vehículo fácilmente y poder realizar cualquier maniobra.<br>2. Los valores que aparecen en el HUD del vídeo deben corresponder con las acciones de giro y velocidad ejecutadas por el operario.<br>3. Las líneas de la carretera deben ser visibles tanto en partes rectas como en partes curvas del recorrido. |
| **Resultado obtenido** | Se obtiene el resultado esperado. |

Tabla 6.5: P-V/SC-02

| **P-V/SC-02** | |
|---|---|
| **Objetivo** | Comprobar que la funcionalidad de velocidad constante mantiene la velocidad en un recorrido variado. |
| **Requisito(s)** | RF-03 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Activar la funcionalidad de velocidad mínima del vehículo.<br>3. Realizar un recorrido que implique llevar la dirección del vehículo a sus valores límite (máximo giro a ambos lados). |
| **Resultado esperado** | La velocidad del vehículo debería ser constante en todo momento o, al menos, el vehículo no se debería quedar parado por falta de torque. |
| **Resultado obtenido** | El vehículo no se queda parado en ningún momento. En algunos casos se puede observar como la velocidad del vehículo aumenta ligeramente al tomar una curva. Esto ocurre debido a que la funcionalidad no es totalmente precisa. |

Tabla 6.6: P-V/SC-03

| **P-V/SC-03** | |
|---|---|
| **Objetivo** | Comprobar que el sistema de control permite guardar datos de entrenamiento. |
| **Requisito(s)** | RF-04 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador. En el ordenador se debe iniciar la versión que guarda datos de entrenamiento.<br>2. Realizar un recorrido variando la velocidad y el giro. |

| | |
|---|---|
| **Resultado esperado** | Se esperan las imágenes del vídeo en tiempo real guardadas con un nombre identificativo y un fichero que relacione las imágenes mediante su identificador con el valor correspondiente a la dirección. |
| **Resultado obtenido** | Se obtiene el resultado esperado. |

Tabla 6.7: P-V/SC-04

| **P-V/SC-04** | |
|---|---|
| **Objetivo** | Comprobar el realismo del control del vehículo (I). |
| **Requisito(s)** | RNF-07 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Utilizar el mando de Xbox One como dispositivo de control.<br>3. Realizar un recorrido. |
| **Resultado esperado** | Utilizando este tipo de control, el operario podría realizar cambios de dirección muy bruscos, un comportamiento no deseado. |
| **Resultado obtenido** | Se obtiene el resultado esperado.<br>La posibilidad de cambiar de dirección con un solo movimiento en el mando de control implica giros muy bruscos. |

Tabla 6.8: P-V/SC-05

| **P-V/SC-05** | |
|---|---|
| **Objetivo** | Comprobar el realismo del control del vehículo (II). |
| **Requisito(s)** | RNF-07 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Utilizar el kit Logitech G29 como control.<br>3. Realizar un recorrido. |
| **Resultado esperado** | Utilizando este tipo de control, el control de la dirección debería ser más suave. |
| **Resultado obtenido** | Utilizando el volante de conducción se consigue girar la dirección del vehículo de forma más suave.<br>Para mover la dirección del vehículo es necesario pasar por los valores intermedios del volante, haciendo que el giro no sea tan brusco. |

Tabla 6.9: P-V/SC-06

| **P-V/SC-06** | |
|---|---|
| **Objetivo** | Comprobar que el vehículo es capaz de tomar curvas de hasta 90º con un radio de, al menos, la longitud del vehículo. |
| **Requisito(s)** | RNF-08 |
| **Procedimiento** | 1. Construir una curva de 90º cuyo radio sea igual a la longitud del vehículo<br>2. Utilizando el sistema de control, tratar de tomar la curva. |
| **Resultado esperado** | El vehículo debería ser capaz de trazar dicha curva sin problema. |
| **Resultado obtenido** | Se obtiene el resultado esperado. |

Tabla 6.10: P-V/SC-07

| P-V/SC-07 | |
|---|---|
| **Objetivo** | Comprobar los siguientes aspectos del vídeo en tiempo real:<br>1. El vídeo es estable.<br>2. Ofrece un mínimo de 10 FPS.<br>3. Tiene una latencia máxima de 0,1 segundos. |
| **Requisito(s)** | RNF-09 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Realizar un recorrido para comprobar que el vídeo es estable.<br>3. Modificar el sistema de visualización para contar los FPS que ofrece.<br>4. Cálculo de la latencia: Situar un cronometro en marcha delante de la cámara del vehículo y tomar una foto del cronómetro y del vídeo en tiempo real. |
| **Resultado esperado** | El vídeo debería ser estable, ofrecer como mínimo 10 FPS y tener una latencia máxima de 0,1 segundos. |
| **Resultado obtenido** | Se obtienen 25 FPS de media y una latencia entre 0,09 y 0,14 segundos. Aunque no se obtiene un valor de latencia máxima inferior al especificado, este es más que suficiente para realizar sin problema la conducción. |

Tabla 6.11: P-V/SC-08

| P-V/SC-08 | |
|---|---|
| **Objetivo** | Comprobar que el vehículo es funcional durante un mínimo de 30 minutos. (Tiempo de operación) |
| **Requisito(s)** | RNF-10 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Realizar un recorrido durante 30 minutos.<br>3. Justo antes de alcanzar los 30 minutos se repite la prueba P-V/SC-07 para comprobar el funcionamiento del vídeo en tiempo real. |
| **Resultado esperado** | El sistema de control y el vehículo deben funcionar correctamente durante el tiempo de operación. Las respuestas mecánicas del vehículo deben ser consistentes durante este periodo de tiempo. |
| **Resultado obtenido** | Se obtiene el resultado esperado. |

Tabla 6.12: P-V/SC-09

| P-V/SC-09 | |
|---|---|
| **Objetivo** | Comprobar que el sistema de control es funcional cuando la distancia hasta el vehículo es de 5 metros (espacio de operación). |
| **Requisito(s)** | RNF-11 |
| **Procedimiento** | 1. Iniciar el sistema de control en el vehículo y en el ordenador.<br>2. Realizar un recorrido con una separación de 5 metros hasta el ordenador en el que se ejecuta el sistema de control. |
| **Resultado esperado** | El sistema de control y el vehículo deben funcionar correctamente en el espacio de operación. |

| | |
|---|---|
| **Resultado obtenido** | El control del vehículo, giro y velocidad, se pueden controlar adecuadamente. Sin embargo, el vídeo no es muy estable a esta distancia.<br>De todas formas, la distancia máxima finalmente necesaria entre ordenador de control y vehículo es de entre 3 y 4 metros. A esta distancia el sistema de control al completo funciona correctamente. |

### 6.1.3. Pruebas referentes al sistema de conducción autónoma

En este apartado se describirá el plan de pruebas elaborado para validar los requisitos relacionados con redes de neuronas y el agente desarrollado.

Tabla 6.13: P-SCA-01

| **P-SCA-01** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente y el cumplimiento de las siguientes restricciones:<br>1. La única entrada al agente es una imagen.<br>2. La tarea de extracción de características de líneas de la carretera la realiza un *Autoencoder*.<br>3. La tarea de control autónomo la implementa una red que usa únicamente las características de las líneas y controla la dirección del vehículo.<br>4. Todo el procesamiento durante la conducción autónoma se realiza en el vehículo. |
| **Requisito(s)** | RF-01, RNF-01, RNF-02, RNF-03 y RNF-05. |
| **Procedimiento** | Teniendo el agente entrenado siguiendo las restricciones anteriores, se coloca el vehículo en el circuito y se inicia agente desarrollado. |
| **Resultado esperado** | El vehículo debe seguir el recorrido del circuito sin salirse. Además, el estilo de conducción del vehículo debe mostrar características parecidas al entrenamiento realizado. |
| **Resultado obtenido** | Se obtiene el resultado esperado. Se puede observar como el agente imita el comportamiento que ha aprendido en el entrenamiento (trazado de curvas). |

Tabla 6.14: P-SCA-02

| **P-SCA-02** | |
|---|---|
| **Objetivo** | Comprobar que las características dadas por el *Autoencoder* corresponden con las líneas de la carretera. La prueba se debe realizar obteniendo la secuencia de conducción en el escenario objetivo. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | Teniendo el *Autoencoder* entrenado:<br>1. Obtener una secuencia de conducción sobre el circuito en el escenario objetivo.<br>2. Obtener la reconstrucción dada por el *Autoencoder* de todos los *frames* capturados. |
| **Resultado esperado** | En la reconstrucción obtenida solo deben aparecer las líneas de la carretera. De esta forma se puede confirmar que las características del vector latente dado por la red corresponden con las líneas de la carretera. |

| | |
|---|---|
| **Resultado obtenido** | Se obtiene el resultado esperado. La reconstrucción pierde calidad en algunos casos, pero el recorrido se comprende correctamente. |

Tabla 6.15: P-SCA-03

| **P-SCA-03** | |
|---|---|
| **Objetivo** | Comprobar si el agente es capaz de conducir en el sentido para el que entrena. Esta prueba se debe realizar para ambos sentidos. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el vehículo en el circuito e iniciar el programa que hace uso del agente desarrollado. El agente debe estar entenado únicamente en el sentido de la marcha.<br>2. Cuando se haya realizado esta conducción, volver al paso 1 colocando el vehículo en el sentido contrario y utilizar el agente entrenado para ese sentido. |
| **Resultado esperado** | Ambos agentes deberían conducir adecuadamente. |
| **Resultado obtenido** | El agente que recorre el circuito en sentido horario lo realiza adecuadamente siempre.<br>El agente que recorre el circuito en sentido antihorario se sale del recorrido en algunas ocasiones. Esto ocurre en la parte más compleja del circuito, la chicane. Al ser la parte más complicada de abordar se podría esperar alguna complicación de este tipo.<br>Un agente entrenado para abordar ambos sentidos obtiene los mismos resultados que por separado. |

Tabla 6.16: P-SCA-04

| **P-SCA-04** | |
|---|---|
| **Objetivo** | Comprobar si el agente es capaz de conducir en ambos sentidos del circuito habiendo entrenado solo en uno de ellos. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | • Colocar el vehículo en el circuito e iniciar el programa que hace uso del agente desarrollado (sentido horario). El vehículo se debe colocar en el sentido antihorario. |
| **Resultado esperado** | Por la naturaleza del circuito desarrollado, este dispone de giros que son exclusivos para cada sentido. Por ejemplo, en el sentido horario se realizan giros de 90º hacia la derecha principalmente.<br>Teniendo esto en cuenta, el agente debería conducir adecuadamente en las rectas, pero no en las curvas mencionadas, ya que en el sentido antihorario son justo al contrario.<br>La chicane del circuito podría llegar a recorrerla adecuadamente dado que en cada sentido lo único que cambia es el orden de los giros a realizar. |
| **Resultado obtenido** | Se obtienen mejores resultados de lo esperado.<br>Las curvas 90º en sentido contrario al que ha entrenado no las consigue superar. Aunque si se puede observar cómo intenta abordarlas. Al conocer curvas hacia ese lado pero de menor intensidad, gira en ese sentido, aunque no lo suficiente. |

| | |
|---|---|
| | El agente entrenado en un solo sentido consigue recorrer la chicane en ambos sentidos sin problemas. |

Tabla 6.17: P-SCA-05

| **P-SCA-05** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente con cambios leves de iluminación. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el vehículo en el circuito e iniciar el agente desarrollado.<br>2. Generar pequeñas sombras en el tramo que visualiza el agente. |
| **Resultado esperado** | El agente podría responder correctamente ante la aparición de pequeñas sombras. |
| **Resultado obtenido** | Se obtiene el resultado esperado. *Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.18: P-SCA-06

| **P-SCA-06** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente con cambios graves de iluminación. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el vehículo en el circuito e iniciar el agente desarrollado.<br>2. Provocar brillos en el circuito. También se prueba con cambios graves en la fuente de luz. |
| **Resultado esperado** | El agente no ha sido entrenado con casos en los que existen diferentes niveles de iluminación. En este caso el agente podría no funcionar adecuadamente. |
| **Resultado obtenido** | La aparición de brillos provoca una reconstrucción incorrecta en una parte de las líneas, el agente pierde el control en algunas ocasiones.<br>Una disminución grande de la fuente de luz provoca que se reconstruya una imagen totalmente en negro. El agente pierde el control.<br>En caso de aumentar en gran cantidad la iluminación de la fuente de luz (sin generar brillos), el agente funciona con normalidad.<br>*Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.19: P-SCA-07

| **P-SCA-07** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente con cambios leves en el escenario objetivo. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el vehículo en el circuito e iniciar agente desarrollado.<br>2. Añadir elementos en el escenario objetivo, a los lados del circuito. |
| **Resultado esperado** | El agente sigue en el mismo escenario objetivo, en el que las características como el color principal o la iluminación no han cambiado. El agente debería responder correctamente ante cambios leves. |

| | |
|---|---|
| **Resultado obtenido** | Se obtiene el resultado esperado. Tanto la reconstrucción como la conducción autónoma se realiza adecuadamente.<br>*Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.20: P-SCA-08

| **P-SCA-08** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente con cambios graves en el escenario objetivo. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el vehículo en el circuito e iniciar agente desarrollado.<br>2. Añadir elementos en el interior del recorrido. Se prueban casos en los que se tapan parcialmente las líneas y otros casos en los que no se tapan las líneas. |
| **Resultado esperado** | Aunque el agente sigue en el mismo escenario objetivo, este no ha sido entrenado con casos en los que el circuito se ve tan afectado. Se espera una reconstrucción incorrecta y, por tanto, un mal funcionamiento del agente. |
| **Resultado obtenido** | Se obtienen resultados mejores de lo esperado.<br>Cuando los elementos añadidos se encuentran en cualquier parte excepto sobre las líneas, el agente funciona con normalidad. En caso de que los elementos creen sombras graves sobre las líneas, la reconstrucción es incorrecta y se pierde el control.<br>Cuando los objetos tapan las líneas, si lo hacen levemente, la reconstrucción no se ve afectada y el agente funciona con normalidad.<br>En caso de que tapen una gran parte de la(s) línea(s), solo se reconstruye bien una parte de la escena. Si se ha tapado una de las dos líneas, el agente recorre el tramo con normalidad en la mayoría de los casos. Si se han tapado ambas líneas, el agente no tiene con qué guiarse y pierde el control.<br>*Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.21: P-SCA-09

| **P-SCA-09** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente en el escenario objetivo montando el circuito de otra forma (diferente al que ha sido utilizado para entrenar). |
| **Requisito(s)** | RNF-04 y RNF-13 |
| **Procedimiento** | 1. Construir un circuito diferente haciendo uso de los tramos utilizados en el circuito inicial.<br>2. Colocar el vehículo en el circuito montado en la prueba P-SCA-09.<br>3. Comprobar la reconstrucción del *Autoencoder* en el nuevo circuito.<br>4. Iniciar el agente entrenado para ambos sentidos del circuito inicial (ya que el nuevo circuito dispone de giros a ambos lados). |
| **Resultado esperado** | Se espera que las reconstrucción de las líneas de la carretera sean correctas siempre que en la imagen aparezca un tramo único |

| | |
|---|---|
| | completo. Las secciones en las que se unen dos tramos que en entrenamiento no ha visto unidos se podrían reconstruir de forma incorrecta.<br>El agente podría conducir de forma adecuada en algunos tramos. |
| **Resultado obtenido** | Se obtienen resultados mejores de lo esperado.<br>Las reconstrucciones encajan con lo que visualiza el agente en el nuevo circuito, y se consigue conducir de forma correcta.<br>En las reconstrucciones se puede observar cómo reconstruye de forma incorrecta la parte más alejada, reconstruyendo en esa zona lo que ha visto en entrenamiento. Sin embargo, cuando se va acercando se corrige y no hay problemas.<br>*Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.22: P-SCA-10

| **P-SCA-10** | |
|---|---|
| **Objetivo** | Comprobar la reconstrucción del *Autoencoder* en el escenario objetivo en un tramo de carretera que represente un cruce (tramo no visto en entrenamiento). |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Construir un nuevo tramo de carretera que represente un cruce de 4 calles.<br>2. Obtener varios *frames* en este nuevo tramo.<br>3. Obtener la reconstrucción del autoencoder. |
| **Resultado esperado** | Ya que este tramo es totalmente desconocido para la red, se espera un resultado totalmente incorrecto. |
| **Resultado obtenido** | Se obtiene una reconstrucción parcialmente incorrecta. Aunque la reconstrucción no es muy buena, se puede observar como el *Autoencoder* trata de reconstruir algo parecido a dos curvas a cada lado del cruce.<br>Se podría decir que generaliza, ya que construye algo parecido a lo que ve sin haberlo visto de esa forma en entrenamiento.<br>*Contenido visual en el repositorio del proyecto* [39]. |

Tabla 6.23: P-SCA-11

| **P-SCA-11** | |
|---|---|
| **Objetivo** | Comprobar el funcionamiento del agente sobre el circuito inicial pero en un escenario diferente para el que ha sido entrenado. |
| **Requisito(s)** | RNF-04 |
| **Procedimiento** | 1. Colocar el circuito sobre otro escenario.<br>2. Colocar el vehículo sobre el circuito.<br>3. Comprobar reconstrucciones del *Autoencoder* en diferentes partes del circuito en el nuevo escenario.<br>4. Iniciar el agente completo para realizar la conducción. |
| **Resultado esperado** | Ya que el agente ha sido entrenado para un escenario específico, se espera que el agente no funcione del todo correcto en otro. |
| **Resultado obtenido** | Se obtienen resultados mejores de lo esperado.<br>El *Autoencoder* obtiene reconstrucciones algo dañadas, aunque aceptables. En algunos casos se obtienen reconstrucciones |

| | |
|---|---|
| | incorrectas ya que la iluminación de este escenario es diferente a la iluminación original.<br>El agente completo desarrolla una conducción que trata de seguir el recorrido, aunque en muchos casos se sale de este. *Contenido visual en el repositorio del proyecto* [39]. |

## 6.2. Matriz de trazabilidad

En este apartado se muestra la matriz de trazabilidad entre requisitos y pruebas. A partir esta matriz se puede comprobar que todo requisito ha sido evaluado en al menos una prueba. Se puede visualizar esta matriz en la Tabla 6.24.

## 6.3. Conclusiones de las pruebas de validación

En este apartado se van a describir conclusiones sobre los resultados de las pruebas de validación más importantes.

En primer lugar, tanto el escenario *chroma* como el sistema de control y el vehículo han cumplido casi por completo con las especificaciones que habían sido estipuladas en los requisitos del proyecto.

En segundo lugar, se ha alcanzado el objetivo del proyecto. Se ha conseguido desarrollar un agente que sigue un recorrido sin salirse de las líneas. Se han realizado muchas pruebas que definen el alcance del agente autónomo desarrollado.

Tal como se indicaba en el requisito RNF-04, se ha evaluado la robustez y generalización del sistema de conducción autónoma. Los resultados obtenidos sobre estas pruebas han sido mejores de lo esperado.

El agente desarrollado para realizar la conducción ha mostrado ser bastante robusto ante cambios en el escenario objetivo. A pesar de añadir objetos, ya sea sobre el asfalto o fuera de él, el agente ha funcionado con normalidad. En los casos en los que se han tapado las líneas de carretera, si se ha hecho de forma leve, el agente ha realizado la reconstrucción sin problemas y ha podido mantener el control. En caso de tapar solo una de las líneas de forma grave, el agente ha podido realizar la conducción guiándose por la otra línea. Sin embargo, en caso de tapar ambas líneas el agente ha perdido el control.

En cuanto a la generalización, se ha podido ver como un agente entrenado para un solo sentido del circuito trata de realizar la conducción en el otro sentido. Debido a la naturaleza del circuito, el agente no ha podido completar la conducción en todos los tramos. Se puede señalar que la parte que más llama la atención es la chicane. El agente ha sido capaz de recorrer este tramo en el sentido contrario para el que ha entrenado sin problemas.

Una prueba de generalización mayor ha consistido en construir un circuito diferente al utilizado para entrenar. Debido a que los tramos del circuito son piezas, se ha podido montar un nuevo circuito. Este nuevo circuito dispone de giros cerrados a ambos lados, por lo que se ha utilizado el agente que ha entrenado para ambos sentidos del circuito inicial. El agente ha demostrado ser capaz de reconstruir las líneas de la carretera y seguir el nuevo recorrido sin problema, aun sin haber entrenado para ese recorrido en concreto.

Teniendo en cuenta las pruebas realizadas, se podría decir que el agente desarrollado ha demostrado tener un buen nivel de robustez y generalización.

Tabla 6.24: Matriz de trazabilidad entre pruebas y requisitos.

| | P-E- | | P-V/SC- | | | | | | | | | P-SCA- | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 01 | 02 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 |
| **RF-01** | | | | | | | | | | | | X | | | | | | | | | | |
| **RF-02** | | | X | | | | | | | | | | | | | | | | | | | |
| **RF-03** | | | | X | | | | | | | | | | | | | | | | | | |
| **RF-04** | | | | | X | | | | | | | | | | | | | | | | | |
| **RF-05** | | | X | | | | | | | | | | | | | | | | | | | |
| **RNF-01** | | | | | | | | | | | | X | | | | | | | | | | |
| **RNF-02** | | | | | | | | | | | | X | | | | | | | | | | |
| **RNF-03** | | | | | | | | | | | | X | | | | | | | | | | |
| **RNF-04** | | | | | | | | | | | | | X | X | X | X | X | X | X | X | X | X |
| **RNF-05** | | | | | | | | | | | | X | | | | | | | | | | |
| **RNF-06** | | | X | | | | | | | | | | | | | | | | | | | |
| **RNF-07** | | | | | | X | X | | | | | | | | | | | | | | | |
| **RNF-08** | | | | | | | | X | | | | | | | | | | | | | | |
| **RNF-09** | | | | | | | | | X | | | | | | | | | | | | | |
| **RNF-10** | | | | | | | | | | X | | | | | | | | | | | | |
| **RNF-11** | | | | | | | | | | | X | | | | | | | | | | | |
| **RNF-12** | X | X | | | | | | | | | | | | | | | | | | | | |
| **RNF-13** | | | | | | | | | | | | | | | | | | | | X | | |

# 7. MARCO REGULADOR

En este capítulo se describirán aspectos sobre legislación vigente en relación con el trabajo realizado. En concreto, se detallará la legislación del coche autónomo en España y las licencias de los recursos software utilizados.

## 7.1. Legislación sobre conducción autónoma en España

En la actualidad, la legislación sobre coches autónomos en España no está muy avanzada. De hecho, algunos artículos del Código de Tráfico y Seguridad Vial [47] chocan directamente con algunas de las características de los coches autónomos. Por ejemplo, en el capítulo III de dicha normativa se indica lo siguiente: Artículo 17.1, «Los conductores deberán estar en todo momento en condiciones de controlar sus vehículos […] » [47]. Artículo 18.1, «El conductor de un vehículo está obligado a mantener […] la atención permanente a la conducción…» [47].

Antes de entrar en mayor detalle, es necesario conocer la taxonomía en la que se clasifican los vehículos en función de su nivel de autonomía. Esta clasificación fue aportada por la SAE, la sociedad de ingenieros de automóviles, y actualmente ha sido adoptada por la mayoría de las organizaciones.

En la Figura 7.1 se pueden observar las características principales de cada categoría sobre la clasificación mencionada.

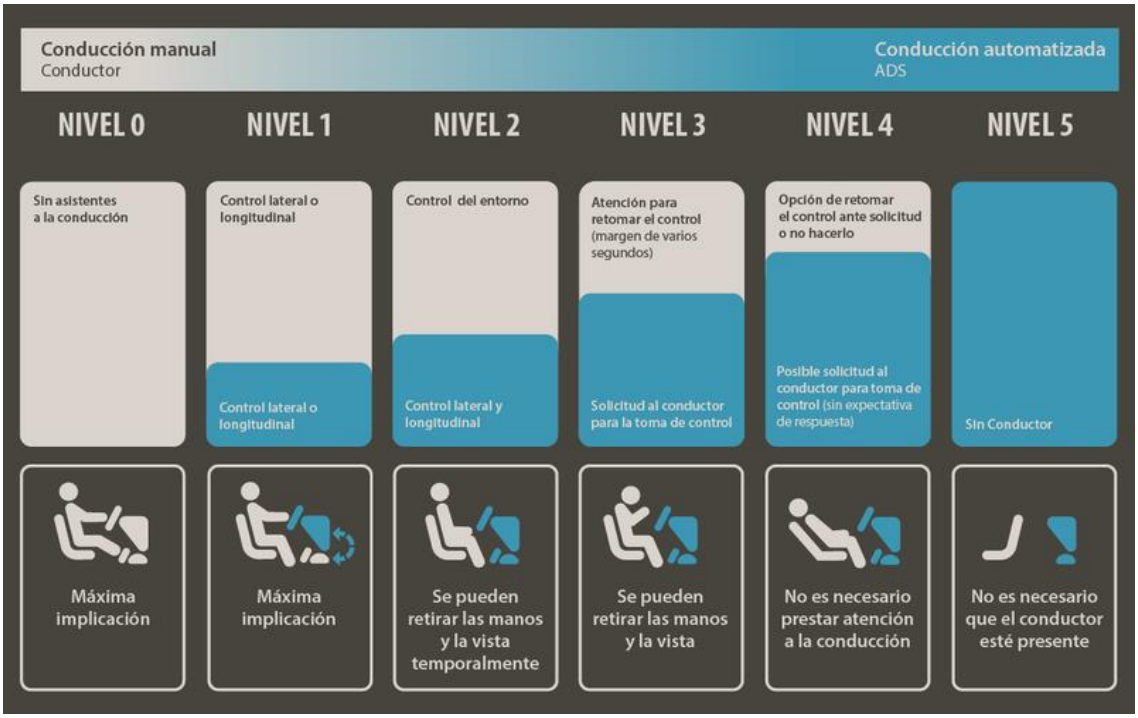



Figura 7.1: Clasificación de vehículos por su nivel de autonomía.

*En cada nivel se marcan las obligaciones de la conducción por parte de un humano (gris) y lo que realiza el vehículo autónomo (azul). En la Tabla I de la instrucción 15/V-113 [48] se puede consultar detalladamente cada nivel de autonomía según la legislación española.*

El vehículo desarrollado en este proyecto quedaría clasificado entre el nivel 1 y 2 de autonomía. En este nivel, el vehículo es capaz de mantener el control lateral o longitudinal tomando información del entorno y el conductor debe tomar el control de la parte de la que no se encarga el sistema. En este proyecto el vehículo controla dinámicamente la

dirección y establece una velocidad constante. Si se controlase la velocidad de forma dinámica, pertenecería al nivel 2 de autonomía.

Según la normativa española actual, se permite la circulación de vehículos autónomos (con funciones activadas) hasta el nivel 3, ya que es necesario que el conductor esté en todo momento en condiciones de controlar su vehículo.

Actualmente, la única normativa española que trata con aspectos de conducción autónoma es una instrucción dada por la DGT, concretamente la Instrucción 15/V-113 [48]. A continuación, se mencionarán los aspectos principales de dicha instrucción.

En primer lugar, en la instrucción se realizan las siguientes definiciones:

- "**VEHÍCULO AUTÓNOMO:** Todo vehículo con capacidad motriz equipado con tecnología que permita su manejo o conducción sin precisar la forma activa de control o supervisión de un conductor, tanto si dicha tecnología autónoma estuviera activada o desactivada, de forma permanente o temporal."
- "**MODO AUTÓNOMO:** modalidad de conducción consistente en el manejo o conducción del vehículo autónomo sin el control activo de un conductor cuando su tecnología autónoma está activada."
- "**MODO CONVENCIONAL:** modalidad de conducción de un vehículo autónomo en la que la tecnología autónoma está desactivada y su conducción o manejo debe efectuarse mediante el control activo de un conductor."

Esta instrucción tiene como objetivo la regulación de la concesión de autorizaciones para la realización de pruebas y ensayos de investigación con vehículos autónomos en vías abiertas al tráfico general. Se publica en orden a garantizar las posibles mejoras que podrían aportar las pruebas y ensayos a la seguridad vial y a la movilidad segura y sostenible en España.

Estas autorizaciones se conceden bajo una serie de requisitos. A continuación se describe alguno de ellos:

- El vehículo debe ser un vehículo autónomo en los términos previstos en la instrucción.
- Las autorizaciones deben ser solicitadas por fabricantes, laboratorios, universidades y consorcios profesionales.
- El conductor del vehículo autónomo será en todo momento el responsable de la conducción y manejo del vehículo.
- El conductor debe estar preparado en todo momento para tomar el control, tanto si se realiza desde el interior del vehículo como si lo conduce remotamente.

El ámbito de la autorización es nacional, pero se deben establecer previamente los tramos de vía urbana e interurbana en los que se realizarán las pruebas. La duración de la instrucción tiene un plazo máximo de 2 años, aunque se puede prorrogar sucesivamente.

Esta instrucción entró en vigor con fecha del 13 de noviembre de 2015. Hasta el día de hoy este es el documento principal sobre conducción autónoma en España. La legislación sobre esta tecnología no está muy avanzada, y será necesario que evolucione de forma que permita una integración más rápida del vehículo autónomo en la sociedad.

El vehículo desarrollado en este proyecto se excluye de los términos de la instrucción mencionada dado que es un prototipo a escala reducida.

Desde la entrada en vigor de dicha instrucción, la DGT solo ha publicado otra instrucción relacionada con la conducción automática. Esta es la INSTRUCCIÓN 16 TV/89 [49], que regula los sistemas de aparcamiento asistido.

## 7.2. Licencias de recursos utilizados

A lo largo del proyecto se han utilizado algunos recursos que disponen de una licencia de uso. A continuación se detallan las licencias de cada uno de ellos.

- MNIST [21]: El conjunto de datos de MNIST está disponible bajos los términos de una licencia de Creative Commons Atribución-CompartirIgual 3.0 (CC BY-SA 3.0) [50].

  Bajo esta licencia es posible compartir el material en cualquier medio o formato y adaptarlo para cualquier propósito. Es necesario dar crédito al autor y en caso de adaptarlo y compartirlo, ese contenido debe estar bajo la misma licencia del original.

- Sign Language MNIST [42]: Este conjunto de datos está disponible bajo una licencia Universal Dedicación de Dominio Público (CC0 1.0) [51].

  CC0 1.0: “La persona que asoció una obra con este resumen ha dedicado la obra al dominio público, mediante la renuncia a todos sus derechos a la obra bajo las leyes de derechos autorales en todo el mundo, incluyendo todos los derechos conexos y afines, en la medida permitida por la ley.” [51]

- Automold [45]: Los ficheros de este programa o repositorio están disponibles también bajo la licencia del MIT. Concretamente con el siguiente titular de derechos de autor: “Copyright (c) 2018 Ujjwal Saxena”.

  Los términos de la licencia del MIT permiten la modificación, distribución y uso del software. Se pueden consultar todos los detalles de la licencia accediendo a la página web que la contiene [52].

# 8. ENTORNO SOCIOECONÓMICO

En este capítulo se describirá el impacto económico y social de la tecnología tratada en este proyecto. También se incluye la planificación y el presupuesto del proyecto.

## 8.1. Impacto socioeconómico

Los vehículos autónomos se están convirtiendo en una tecnología de nuestro presente. Cada día son más las empresas involucradas en lo que aparenta ser uno de los avances del siglo. Empresas como Google, Amazon o Nvidia, del ámbito tecnológico, desarrollan grandes proyectos sobre conducción autónoma. También empresas automovilísticas, como Audi, Mercedes o Toyota, están evolucionando y trabajan en ello.

Actualmente existen vehículos comerciales que se aproximan al nivel 3 de autonomía y que están disponibles al consumidor. Un ejemplo de este caso podría ser la empresa Tesla [3] que, además, ya equipa a sus vehículos con la toda tecnología necesaria para para ofrecer en un futuro próximo la capacidad de conducción autónoma total. Otra de las grandes conocidas es Waymo, una empresa de Google que ya tiene vehículos autónomos de nivel 4 y que realiza pruebas en zonas públicas delimitadas en EEUU.

Es importante señalar las grandes ventajas que los vehículos autónomos pueden aportar a la sociedad. En primer lugar, una de las ventajas más llamativas e importantes tiene que ver con los accidentes de tráfico. En torno al 90% de estos accidentes son causados por errores humanos. Los sistemas autónomos podrían eliminar situaciones de exceso de velocidad, adelantamientos poco adecuados, alcohol al volante… reduciendo en gran medida la cantidad de accidentes. Esta ventaja aporta beneficios económicos y sociales, dado que se reducirían gastos, por ejemplo, en el servicio de salud, asistencia en carretera, y equipos de emergencias entre otros. También se reduciría la mortalidad en carretera y otras afecciones que traen como consecuencia los accidentes.

Existen otras ventajas, como el desplazamiento sencillo para personas con movilidad reducida o la eficiencia en el consumo, teniendo como resultado una reducción de gastos económicos y de emisiones contaminantes. Además, la llegada de esta tecnología suele llegar acompañada de otros avances tecnológicos, como la aparición del vehículo eléctrico, también positivo para reducir la contaminación. A nivel económico y social se puede señalar un aumento de la productividad de las personas a causa del tiempo que se podría aprovechar en el transporte, sin necesidad de prestar atención en la conducción.

Aunque la tecnología avanza rápido, la legislación no lo hace tanto. Como ya se ha comentado en el capítulo anterior, las regulaciones relacionadas con esta tecnología aún no están muy avanzadas y están frenando la llegada del coche autónomo.

Sin embargo, antes de salir al mercado es necesario que la tecnología madure lo suficiente y no cause más problemas de los que resuelva. Para ello algunas técnicas utilizadas en estos sistemas necesitan de entrenamiento a partir de una gran cantidad de datos. Por ejemplo, los vehículos de Tesla recogen datos constantemente con el objetivo de utilizarlos para entrenar y mejorar sus sistemas. Otro de los grandes avances relacionados con esta tecnología es el coche conectado, que permite que los vehículos puedan comunicarse entre ellos y tomar decisiones conjuntas.

El aprendizaje de la conducción mostrado en este proyecto se realiza mediante aprendizaje supervisado. Los sistemas que aprenden de esta forma utilizan datos dados por un conductor, como el giro, la aceleración o la frenada, de forma que el vehículo aprende a conducir imitando el comportamiento del conductor.

Un sistema de conducción que utiliza aprendizaje supervisado se podría integrar en los vehículos autónomos previamente entrenados para controlar la dirección y velocidad. Este podría ofrecer diferentes estilos de conducción, más suaves o agresivos dentro de lo permitido. Incluso podría ser viable crear perfiles propios que modifiquen ligeramente ese entrenamiento inicial y se adapten al estilo del conductor. De esta forma, sería posible hacer que el propietario del vehículo o los ocupantes de este se sientan más seguros y cómodos.

## 8.2. Planificación

Este proyecto se ha desarrollado bajo el marco de una Beca de Colaboración concedida por el Ministerio de Educación y Formación Profesional de España en la convocatoria 2019/2020. La concesión de esta beca implica una serie de condiciones a cumplir durante el proyecto por parte del beneficiario. Una de las condiciones de la concesión implica un trabajo diario de 3 horas durante 8 meses, sumando un total de 480 horas de trabajo.

Este proyecto se ha estructurado en fases o agrupaciones de tareas. De esta forma se han abordado todas las tareas necesarias de forma incremental en dificultad, facilitando en gran medida el desarrollo. No todas las fases son dependientes o necesariamente secuenciales. Algunas fases se han desarrollado simultáneamente en el tiempo y otras dependen de la finalización de una fase previa.

Durante la finalización del curso 2018/2019 se tiene el primer contacto con el tutor para realizar el proyecto. En ese momento no se plantea un objetivo concreto, pero se indica que se podría trabajar con robótica y con *Deep Learning*. Se realiza una aproximación inicial a estos temas antes del comienzo del curso 2019/2020. Estas tareas se han agrupado en la fase 0 del proyecto.

Durante las primeras dos semanas del curso se plantean ideas algo más centradas sobre la temática del proyecto. Una vez se tiene una idea medianamente clara, se comienzan simultáneamente las dos primeras fases (fase 1 y 2). La fase 1 trata con el seguimiento de objetos utilizando visión artificial y la fase 2 trata con el aprendizaje de D*eep Learning*. Cuando ya se ha tratado con algunos aspectos principales, la fase 2 queda en segundo plano. En la fase 3 se desarrolla el sistema de control y de obtención de datos de entrenamiento, al finalizar el desarrollo se continúa trabajando con *Deep Learning*, esta vez más orientado al objetivo del proyecto. Una vez se realiza una aproximación a la tarea de extracción de líneas en un simulador, se comienza la fase 4, en la que se desarrolla el primer entorno de trabajo simplificado. Cuando se confirma la viabilidad de dicha tarea, se comienza la fase 5, en la que se desarrollan las tareas principales en el entorno final. Se termina con la fase 5 a comienzos del mes de abril y se comienza con el desarrollo de la memoria del proyecto, que finaliza en el mes de julio de 2020.

Debido al retraso de la fecha de entrega ocasionado por la COVID-19, la etapa de documentación se ha prolongado. Inicialmente se esperaba finalizar con la memoria en el mes de mayo. El tiempo adicional ha sido aprovechado para mejorar la documentación, las pruebas de validación y producir vídeos que facilitasen la comprensión del trabajo realizado.

El proyecto comienza en el mes de septiembre de 2019 y acaba en el mes de julio de 2020, teniendo una duración de unos 11 meses en los que se han invertido unas 750 horas de trabajo. Teniendo en cuenta lo anterior, se podría decir que durante el desarrollo del proyecto se han invertido aproximadamente 3 horas y media diarias, suponiendo 5 días de trabajo a la semana.

En la Tabla 8.1 se muestra un desglose detallado de las tareas realizadas en cada fase del proyecto, junto a su fecha de inicio y finalización y el tiempo dedicado en horas.

Tabla 8.1: Tareas realizadas durante el desarrollo del proyecto.

| Tarea | Fecha inicio | Fecha fin | Duración (horas) |
|---|---|---|---|
| **Fase 0: Trabajo Previo** | **15/08/2019** | **14/09/2019** | **20** |
| Aproximación a la electrónica con Arduino | 15/08/2019 | 15/08/2019 | 2 |
| Montaje y estudio del vehículo inicial | 16/08/2019 | 20/08/2019 | 5 |
| Estudio inicial sobre *Deep Learning* | 21/08/2019 | 31/08/2019 | 5 |
| Aproximación a visión artificial | 21/08/2019 | 31/08/2019 | 5 |
| Planteamiento Inicial | 01/09/2019 | 14/09/2019 | 3 |
| **Fase 1: Seguimiento de objetos con OpenCV** | **15/09/2019** | **28/09/2019** | **14** |
| Análisis | 15/09/2019 | 21/09/2019 | 7 |
| Diseño | 22/09/2019 | 26/09/2019 | 4 |
| Experimentación | 27/09/2019 | 28/09/2019 | 3 |
| **Fase 2: Experimentación con *Deep Learning* (I)** | **15/09/2019** | **30/11/2019** | **100** |
| Estudio de Deep Learning | 15/09/2019 | 27/09/2019 | 25 |
| Dogs vs Cats | 28/09/2020 | 17/11/2019 | 45 |
| Sign Language MNIST | 18/11/2019 | 30/11/2019 | 30 |
| **Fase 3: Sistema de control y etiquetado** | **30/11/2019** | **09/01/2020** | **71** |
| Análisis | 30/11/2019 | 30/12/2019 | 26 |
| Diseño e implementación | 05/12/2019 | 09/01/2020 | 45 |
| **Fase 2: Experimentación con *Deep Learning* (II)** | **10/01/2020** | **18/02/2019** | **91** |
| MNIST – Autoencoders | 10/01/2020 | 01/02/2020 | 39 |
| Extracción de líneas – Simulador | 02/02/2020 | 18/02/2020 | 52 |
| **Fase 4: Extracción de líneas de carretera (Escenario I)** | **19/02/2020** | **11/03/2020** | **67** |
| Análisis del problema | 19/02/2020 | 24/02/2020 | 10 |
| Diseño | 20/02/2020 | 26/02/2020 | 7 |
| Experimentación | 27/02/2020 | 11/03/2020 | 50 |
| **Fase 5: Diseño y experimentación final** | **12/03/2020** | **13/04/2020** | **97** |
| Análisis del problema | 12/03/2020 | 13/03/2020 | 3 |
| Diseño | 14/03/2020 | 19/03/2020 | 24 |
| Experimentación | 20/03/2020 | 13/04/2020 | 70 |
| **Validación** | **01/05/2020** | **15/05/2020** | **30** |
| **Documentación** | **14/04/2020** | **02/07/2020** | **260** |
| Redacción de la memoria | 14/04/2020 | 02/07/2020 | 220 |
| Producción de vídeos | 15/05/2020 | 15/06/2020 | 40 |
| **Total** | | | **750** |

En la Figura 8.1 se puede observar el diagrama de Gantt que representa la planificación final del proyecto.

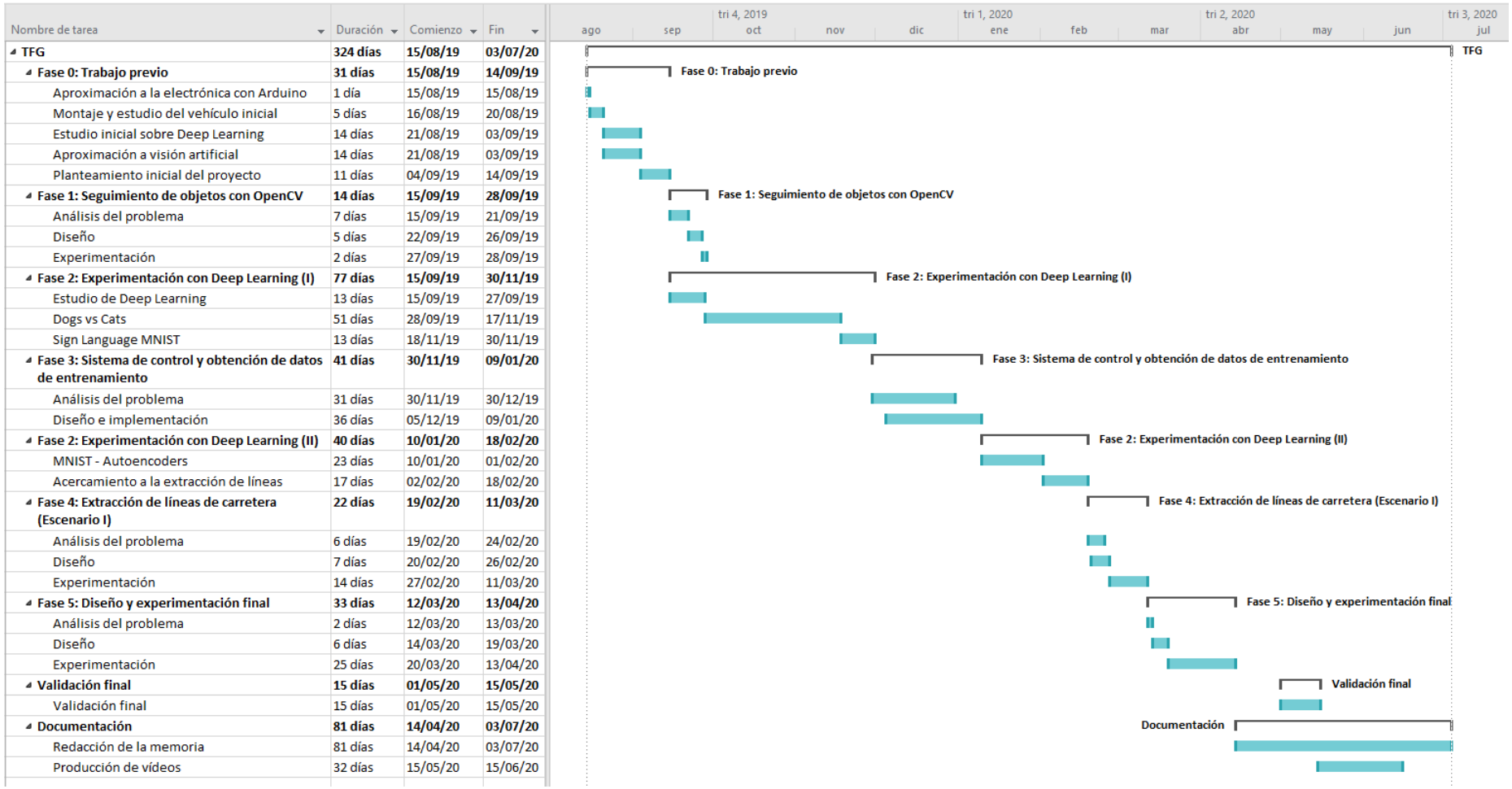



Figura 8.1: Diagrama de Gantt del proyecto.

## 8.3. Presupuesto

Al igual que cualquier otro proyecto, ha sido necesario adquirir el material y contratar la mano de obra necesaria para abordar las tareas.

En este apartado se especifican los costes asociados a la elaboración del proyecto. Se han tenido en cuenta costes de recursos humanos, materiales y licencias software.

### 8.3.1. Costes de recursos humanos

Los costes de recursos humanos tienen en cuenta los salarios de los trabajadores involucrados en el proyecto. En este caso se han tenido dos trabajadores, Javier Corrochano Jiménez, autor del trabajo; y Juan Manuel Alonso Weber, tutor del trabajo.

Javier ha desarrollado tareas correspondientes con diferentes puestos dentro de un proyecto, como analista, diseñador, programador o técnico de pruebas. Sin embargo, se tendrá en cuenta un puesto de ingeniero junior con experiencia inferior a un año. Siguiendo el informe Infoempleo Adecco 2018 [53], el salario medio anual de un ingeniero junior con una experiencia inferior a 1 año es de 22.013,46 €. Teniendo en cuenta una jornada a tiempo completo y un total de 1.920 horas de trabajo anual, sin tener en cuenta festivos y vacaciones, el salario podría ser de unos 11,5 €/hora.

Juan Manuel ha desarrollado tareas relacionadas con la coordinación y supervisión del proyecto, por lo que se podría tener en cuenta un puesto directivo. Siguiendo el informe Infoempleo Adecco 2018, el salario medio anual que percibe un profesional en un puesto directivo es de 43.237,38 €. Teniendo en cuenta una jornada a tiempo completo y un total de 1.920 horas de trabajo anual, el salario sería de unos 22,5 €/hora.

En la Tabla 8.2 se muestran los costes asociados a los salarios de los trabajadores. Los costes relacionados con impuestos se tendrán en cuenta en los costes totales del proyecto.

Tabla 8.2: Costes de recursos humanos.

| Empleado | Horas de trabajo | Coste por hora | Coste en el proyecto |
|---|---|---|---|
| Alonso Weber, Juan Manuel | 80 horas | 22,50 € | 1.800 € |
| Corrochano Jiménez, Javier | 750 horas | 11,50 € | 8.625 € |
| **Total** | | | **10.425 €** |

### 8.3.2. Costes de recursos materiales

Los costes de recursos materiales tienen en cuenta los costes asociados con el hardware, licencias software de pago y otros materiales utilizados para desarrollar el proyecto. Se muestra el coste bruto del producto, es decir, sin incluir impuestos. En los costes totales se indicará el coste incluyendo impuestos.

La vida útil de los recursos utilizados no se limita al proyecto, por lo que se tendrá en cuenta un coste proporcional al tiempo que han sido necesarios. Los productos que se limitan a la duración del proyecto no tendrán en cuenta este aspecto.

Algunas licencias, como Windows 10 o el paquete Office 365, no han supuesto costes ya que la Universidad las proporciona de forma gratuita a estudiantes.

Los costes materiales se han desglosado en coste de herramientas, el material del prototipo final, material de experimentación y de licencias de Software. Se muestran en la Tabla 8.3, Tabla 8.4, Tabla 8.5, Tabla 8.6 y Tabla 8.7.

Tabla 8.3: Coste de herramientas.

| Producto | Uds. | Coste bruto | Uso (meses) | Vida útil (meses) | Coste (sin IVA) |
|---|---|---|---|---|---|
| Ordenador portátil | 1 | 743 € | 10 | 72 | 103,19 € |
| Ordenador remoto para *Deep Learning* | 1 | 632 € | 10 | 72 | 87,78 € |
| Ordenador personal - Documentación y edición de vídeo | 1 | 640 € | 10 | 72 | 88,89 € |
| Teléfono personal | 1 | 206 € | 10 | 24 | 85,83 € |
| Cámara Nikon D3200 | 1 | 413 € | 3 | 120 | 10,33 € |
| Cargador de baterías AA – NiMh | 1 | 13,22 € | 10 | 24 | 5,51 € |
| Volante Logitech G29 Driving Force | 1 | 249 € | 7 | 60 | 29,05 € |
| **Total de Herramientas** | | | | | **410,58 €** |

Tabla 8.4: Coste de material del prototipo final.

| Producto | Uds. | Coste bruto | Uso (meses) | Vida útil (meses) | Coste (sin IVA) |
|---|---|---|---|---|---|
| Smart Video Car Kit SUNFOUNDER | 1 | 69,84 € | - | - | 69,84 € |
| Raspberry Pi 3B+ | 1 | 37,32 € | - | - | 37,32 € |
| Pi Camera rev 2.2 | 1 | 21,61 € | - | - | 21,61 € |
| SanDisk MicroSD 32 GB | 1 | 5,40 € | - | - | 5,40 € |
| 8 x Baterías EBL AA NiMh 2800 mAh | 1 | 17,86 € | - | - | 17,86 € |
| Porta pilas vehículo - 8 x AA | 1 | 3,32 € | - | - | 3,32 € |
| TT Motor DC 1:90 | 2 | 5,54 € | - | - | 11,08 € |
| **Total de material del prototipo final** | | | | | **166,43 €** |

Tabla 8.5: Coste de material de experimentación.

| Producto | Uds. | Coste bruto | Uso (meses) | Vida útil (meses) | Coste (sin IVA) |
|---|---|---|---|---|---|
| ELEGOO Kit Medio ES-EL-KIT-003 | 1 | 19,68 € | 1 | 12 | 1,64 € |
| SanDisk MicroSD 32 GB | 1 | 5,38 € | 10 | 24 | 2,24 € |
| 2 x Baterías 18650 Lítio | 1 | 13,88 € | 1 | 6 | 2,31 € |
| 2 x Baterías AA NiMh 1600 mAh | 4 | 2,44 € | - | - | 9,76 € |
| Raspberry Pi 3B | 1 | 28,49 € | 9 | 24 | 10,68 € |
| Cargador de baterías 18650 | 1 | 13,23 € | 1 | 12 | 1,10 € |
| TT Motor DC 1:120 | 2 | 4,71 € | 1 | 12 | 0,79 € |
| DC-DC Step Down LM2596 | 1 | 2,62 € | 1 | 12 | 0,22 € |
| Pi Camera rev 1.3 | 1 | 14,05 € | 5 | 24 | 2,93 € |
| Mando Xbox One | 1 | 38,67 € | 2 | 24 | 3,22 € |
| Material de papelería (escenario de conducción) | 1 | 16,54 € | - | - | 16,54 € |
| **Total** | | | | | **51,44 €** |

Tabla 8.6: Coste de licencias de software

| Producto | Uds. | Coste bruto | Uso (meses) | Vida útil (meses) | Coste (sin IVA) |
|---|---|---|---|---|---|
| Paquete Adobe Creative Cloud Mensual (Photoshop, Illustrator y Premiere) | 3 | 16,34 € | - | - | 49,02 € |
| Fritzing (diseño de circuitos electrónicos) | 1 | 6,68 € | - | - | 6,68 € |
| **Total de licencias de software** | | | | | **55,70 €** |

Tabla 8.7: Coste total de material

| **Total de material** | |
|---|---|
| **Concepto** | **Coste (sin IVA)** |
| Herramientas | 410,58 € |
| Material del prototipo final | 166,43 € |
| Material experimental | 51,44 € |
| Licencias de Software | 55,70 € |
| **Total** | **684,15 €** |

### 8.3.3. Costes totales

Los costes totales tienen en cuenta los costes brutos de material y de recursos humanos, los impuestos y un fondo de prevención destinado a imprevistos del 10%. Los impuestos corresponden con un 21% de IVA para los costes de material y un 19% de IRPF para los costes de recursos humanos.

En la Tabla 8.8 se encuentran los costes totales del proyecto.

Tabla 8.8: Costes totales del proyecto.

| Tipo | Coste bruto | Impuestos | Coste neto |
|---|---|---|---|
| Recursos humanos | 10.425,00 € | 1.980,75 € | 12.406 € |
| Materiales | 684,15 € | 143,67 € | 827,82 € |
| **Total antes de fondo de imprevistos** | | | **13.233,57 €** |
| Fondo para imprevistos del 10 % | | | 1.323,36 € |
| **Costes totales** | | | **14.556,93 €** |

# 9. CONCLUSIONES Y TRABAJOS FUTUROS

En este capítulo se describirán las conclusiones con respecto al trabajo realizado durante todo el proyecto. Además, se incluye una sección en la que se proponen trabajos futuros.

## 9.1. Conclusiones

En definitiva y como conclusión a grandes rasgos, se ha podido observar como el enfoque tratado en este proyecto es una posibilidad funcional para controlar de forma autónoma la dirección de un vehículo a escala reducida.

El enfoque propuesto sobre el uso de un *Autoencoder* ha resultado satisfactorio. Esta técnica ha sido muy útil para reducir la dimensionalidad de la entrada y obtener un conjunto de características que permiten comprender el entorno de forma no explícita.

Se han utilizado más técnicas o variantes de *Autoencoders* hasta obtener una arquitectura equivalente a un *Denoising Stacked Convolutional Autoencoder*. Uno de los aspectos que mayor importancia ha tenido en el proyecto ha sido la adición de ruido sobre los datos de entrada (*Denoising*). Esto ha sido fundamental para que la red entrenada con las imágenes aumentadas funcionase de forma correcta en el escenario objetivo. Se ha comprobado como esta técnica de regularización de redes de neuronas puede ser útil para mejorar la robustez y generalización de una red neuronal.

En relación con lo anterior, utilizar el escenario *chroma* o artificial es lo que ha posibilitado diferenciar las líneas de la carretera del resto de elementos de la escena. De esta forma ha sido posible obtener un vector de características relacionado únicamente con dichas líneas. Además, este escenario ha permitido generar imágenes aumentadas útiles para entrenar al agente y utilizarlo a posteriori en el escenario de conducción (escenario objetivo).

El aprendizaje de la conducción a partir de las características extraídas ha resultado satisfactorio. Se ha podido observar como el agente desarrollado imita el comportamiento de los datos de conducción capturados. Este resultado se ha obtenido al aplicar una técnica de aprendizaje supervisado, la Imitación del Comportamiento (*Behavioral Cloning*). Una conclusión importante que se ha extraído es que el estilo de conducción del entrenamiento tiene bastante importancia. Con una conducción cuidadosa por parte del usuario, el agente resultante ha sido capaz de recorrer el circuito satisfactoriamente.

Además, se han podido aprovechar algunas ventajas del enfoque desarrollado. La fase de aprendizaje se ha dividido en dos tareas, la extracción de características y el aprendizaje de la conducción. Esta división ha permitido que el desarrollo del sistema resulte más flexible, siendo posible abordar el trabajo de forma incremental y reduciendo el tiempo de entrenamiento necesario. Una vez entrenado el *Autoencoder*, entrenar la red encargada de la conducción es un proceso más rápido. De esta forma, es posible entrenar diferentes estilos de conducción con mayor facilidad. Un enfoque basado en redes de neuronas convolucionales hubiese implicado reentrenar toda la red.

En cuanto al sistema de control del vehículo, se ha alcanzado un control con algunos aspectos realistas. Tanto el control manual de la dirección, realizado con un volante, como la visualización de la carretera, realizada desde una vista frontal en tiempo real, reflejan bastante la realidad. Debido a una limitación de espacio el recorrido dispone de curvas muy cerradas, algo no muy típico en una carretera real. Aun así, el agente autónomo ha sido capaz de recorrer dichas curvas.

## 9.2. Trabajos futuros

Basándose en las tareas realizadas, se pueden definir algunas líneas de trabajo que abordar en el futuro. Las siguientes propuestas aumentarían el alcance y las funcionalidades desarrolladas en este proyecto.

- Mejorar la tarea de extracción de características de las líneas de la carretera en el mundo real, haciendo que sea funcional en carreteras realistas. Sería necesario generar conjuntos de datos reales para aislar las líneas. Dado que aislar las líneas de forma manual o usando técnicas de Visión Artificial puede ser tedioso o complejo, se propone utilizar el escenario *Chroma*.

  Haciendo uso del escenario *chroma* desarrollado en el proyecto, pero generando imágenes aumentadas mucho más realistas para que el *Autoencoder* pueda funcionar correctamente en escenas totalmente realistas, es decir, en carreteras reales. También se propone usar un simulador de un entorno realista que permita generar este tipo de imágenes.

- Utilizar material de mejor calidad. Sobre todo en cuanto al hardware, utilizando un procesador con mayor rendimiento. De esta forma sería posible tener un control más seguro en la conducción y aumentar la velocidad de inferencia y del vehículo. Una posibilidad sería utilizar una Nvidia Jetson Nano. En cuanto a los componentes del vehículo se propone utilizar motores de mayor calidad que ofrezcan un buen rango de velocidades, y un eje de dirección más preciso.

- Regular el ángulo de elevación de la cámara permite que el agente pueda visualizar con mayor o menor antelación el recorrido a realizar, y por tanto, que tome las decisiones en diferentes momentos. Este aspecto permite que un agente entrenado a una velocidad determinada se pueda utilizar a diferentes velocidades, modificando dicho ángulo mediante un servomotor. Otra posibilidad podría ser seleccionar franjas de la imagen en función de la velocidad requerida, utilizando una cámara con mayor ángulo de visión.

- Mejorar el realismo del sistema de control del vehículo. Se podría instalar una cámara 360º en el vehículo y utilizar un casco de realidad virtual de forma que el usuario tenga una perspectiva realista de la conducción.

  Aprovechando el realismo alcanzado con esta propuesta, se podría utilizar el escenario artificial para recrear situaciones complejas en escenarios realistas y entrenar al agente autónomo en estos casos.

# 10. BIBLIOGRAFÍA


[1] «Technology», *Waymo*. https://waymo.com/tech/ (accedido jun. 25, 2020).

[2] «Putting Self-Driving Cars On The Roads | General Motors». https://www.gm.com/our-stories/self-driving-cars.html (accedido jun. 25, 2020).

[3] «Piloto automático | Tesla». https://www.tesla.com/es_ES/autopilot?redirect=no (accedido jun. 25, 2020).

[4] «Waymo expands driverless car testing to Florida», *VentureBeat*, ago. 20, 2019. https://venturebeat.com/2019/08/20/waymo-expands-driverless-car-testing-to-florida/ (accedido jul. 02, 2020).

[5] W. S. McCulloch y W. Pitts, «A logical calculus of the ideas immanent in nervous activity», *Bull. Math. Biophys.*, vol. 5, n.º 4, pp. 115-133, dic. 1943, doi: 10.1007/BF02478259.

[6] D. Hebb, «The organization of behavior; a neuropsychological theory», 1949.

[7] F. F. Rosenblatt, «The perceptron: a probabilistic model for information storage and organization in the brain», *Psychol. Rev.*, 1958, doi: 10.1037/h0042519.

[8] M. Minsky y S. Papert, *Perceptrons; an introduction to computational geometry*. Cambridge, Mass.: MIT Press, 1969.

[9] D. E. Rumelhart, G. E. Hinton, y R. J. Williams, «Learning representations by back-propagating errors», *Nature*, vol. 323, n.º 6088, pp. 533-536, oct. 1986, doi: 10.1038/323533a0.

[10] D. B. Parker, *Learning-logic casting the cortex of the human brain in silicon*. Cambridge, Mass: Center for Computational Research in Economics and Management Science, Alfred P. Sloan School of Management, Massachusetts Institute of Technology, 1985.

[11] Y. Le Cun, «Learning Process in an Asymmetric Threshold Network», *Disord. Syst. Biol. Organ.*, pp. 233-240, 1986, doi: 10.1007/978-3-642-82657-3_24.

[12] G. Hinton, S. Osindero, y Y.-W. Teh, «A Fast Learning Algorithm for Deep Belief Nets», *Neural Comput.*, vol. 18, pp. 1527-54, ago. 2006, doi: 10.1162/neco.2006.18.7.1527.

[13] Y. Bengio, P. Lamblin, D. Popovici, y H. Larochelle, «Greedy Layer-Wise Training of Deep Networks», *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, y T. Hoffman, Eds. MIT Press, 2007, pp. 153–160.

[14] M. A. Ranzato, C. Poultney, S. Chopra, y Y. L. Cun, «Efficient Learning of Sparse Representations with an Energy-Based Model», *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, y T. Hoffman, Eds. MIT Press, 2007, pp. 1137–1144.

[15] «A.M. Turing Award Winners by Year». https://amturing.acm.org/byyear.cfm (accedido jun. 28, 2020).

[16] A. Krizhevsky, I. Sutskever, y G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, y K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[17] K. Hornik, M. Stinchcombe, y H. White, «Multilayer feedforward networks are universal approximators», *Neural Netw.*, vol. 2, n.º 5, pp. 359-366, 1989, doi: https://doi.org/10.1016/0893-6080(89)90020-8.

[18] F. Berzal, *Redes Neuronales & Deep Learning*. Granada: Independiente, 2018.

[19] F. Chollet, *Deep Learning with Python*, First Ed. Shelter Island, New York: Manning Publications, 2017.

[20] D. J. Torres, *DEEP LEARNING Introducción práctica con Keras*. Independently published, 2018.

[21] Y. LeCun y C. Cortes, «MNIST handwritten digit database», 2010, Accedido: abr. 14, 2020. [En línea]. Disponible en: http://yann.lecun.com/exdb/mnist/.

[22] J. Sietsma y R. J. F. Dow, «Creating artificial neural networks that generalize», *Neural Netw.*, vol. 4, n.º 1, pp. 67-79, ene. 1991, doi: 10.1016/0893-6080(91)90033-2.

[23] R. Atienza, *Advanced Deep Learning with Keras*, First Ed. Packt Publishing, 2018.

[24] P. Vincent, H. Larochelle, Y. Bengio, y P.-A. Manzagol, «Extracting and composing robust features with denoising autoencoders», *Proceedings of the 25th international conference on Machine learning*, Helsinki, Finland, jul. 2008, pp. 1096–1103, doi: 10.1145/1390156.1390294.

[25] V. Badrinarayanan, A. Kendall, y R. Cipolla, «SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation», *ArXiv151100561 Cs*, oct. 2016, Accedido: jun. 03, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1511.00561.

[26] D. A. Pomerleau, «ALVINN: An Autonomous Land Vehicle in a Neural Network», *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313.

[27] M. Williams, «PROMETHEUS-The European research programme for optimising the road transport system in Europe», *IEE Colloquium on Driver Information*, 1988, p. 1/1-1/9.

[28] M. Bojarski *et al.*, «End to End Learning for Self-Driving Cars», *ArXiv160407316 Cs*, abr. 2016, Accedido: abr. 24, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1604.07316.

[29] Y. Yang, Z. Wu, Q. Xu, y F. Yan, «Deep Learning Technique-Based Steering of Autonomous Car», *Int. J. Comput. Intell. Appl.*, vol. 17, n.º 02, p. 1850006, may 2018, doi: 10.1142/S1469026818500062.

[30] D. Hearn y M. P. Baker, *Graficos Por Computadora Con Opengl*, Tercera edición. Madrid etc.: ALHAMBRA, 2005.

[31] «Keras vs TensorFlow vs PyTorch | Deep Learning Frameworks», *Edureka*, dic. 05, 2018. https://www.edureka.co/blog/keras-vs-tensorflow-vs-pytorch/ (accedido abr. 20, 2020).

[32] Martín Abadi *et al.*, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[33] F. Chollet y others, *Keras*. 2015.

[34] A. Paszke *et al.*, «PyTorch: An Imperative Style, High-Performance Deep Learning Library», *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, y R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035.

[35] The Theano Development Team *et al.*, «Theano: A Python framework for fast computation of mathematical expressions», *ArXiv160502688 Cs*, may 2016, Accedido: jun. 23, 2020. [En línea]. Disponible en: http://arxiv.org/abs/1605.02688.

[36] «TensorFlow 2.0 is now available!» https://blog.tensorflow.org/2019/09/tensorflow-20-is-now-available.html (accedido jun. 23, 2020).

[37] «Keras 2.3.0. Release», *GitHub*. https://github.com/keras-team/keras (accedido jun. 23, 2020).

[38] G. Bradski, «The OpenCV Library», *Dr Dobbs J. Softw. Tools*, 2000.

[39] J. Corrochano Jiménez, «javiercorrochano/TFGAutonomousCar», *TFGAutonomousCar*. https://github.com/javiercorrochano/TFGAutonomousCar (accedido jun. 27, 2020).

[40] A. Rosebrock, «Ball Tracking with OpenCV», *PyImageSearch*, sep. 14, 2015. https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/ (accedido jun. 27, 2020).

[41] «Dogs vs. Cats». https://kaggle.com/c/dogs-vs-cats (accedido abr. 29, 2020).

[42] «Sign Language MNIST». https://kaggle.com/datamunge/sign-language-mnist (accedido abr. 29, 2020).

[43] «Fashion MNIST». https://kaggle.com/zalando-research/fashionmnist (accedido abr. 29, 2020).

[44] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, y A. Oliva, «Learning Deep Features for Scene Recognition using Places Database», *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, y K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 487–495.

[45] UjjwalSaxena, «UjjwalSaxena/Automold--Road-Augmentation-Library». https://github.com/UjjwalSaxena/Automold--Road-Augmentation-Library (accedido may 15, 2020).

[46] «¿Qué es un Chroma Key?», *Monsuton*, ago. 29, 2019. https://www.monsuton.com/croma-key/ (accedido jul. 02, 2020).

[47] «BOE.es - Código de Tráfico y Seguridad Vial». https://www.boe.es/biblioteca_juridica/codigos/codigo.php?id=020_Codigo_de_Trafico_y_Seguridad_Vial&modo=1 (accedido jun. 27, 2020).

[48] «INSTRUCCIÓN 15/V-113», Subdirección General de la Gestión de la Movilidad, nov. 2015.

[49] «INSTRUCCIÓN 16/TV-89», Subdirección General de la Gestión de la Movilidad, ene. 2016.

[50] «Creative Commons — Attribution-ShareAlike 3.0 Unported — CC BY-SA 3.0». https://creativecommons.org/licenses/by-sa/3.0/deed.en (accedido jun. 27, 2020).

[51] «Creative Commons — CC0 1.0 Universal». https://creativecommons.org/publicdomain/zero/1.0/deed.es (accedido jun. 07, 2020).

[52] «The MIT License | Open Source Initiative». https://opensource.org/licenses/MIT (accedido jun. 07, 2020).

[53] «Informe Infoempleo Adecco 2018», *Infoempleo*. https://www.infoempleo.com/informe-infoempleo-adecco/ (accedido jun. 27, 2020).

# 11. VIDEOGRAFÍA

En este capítulo se incluye el nombre y descripción de los vídeos del proyecto. Los vídeos se encuentran en un repositorio alojado en GitHub con el nombre de **TFGAutonomousCar**. La dirección a dicho repositorio es **https://github.com/javiercorrochano/TFGAutonomousCar**.

## 11.1. Vídeos de la Fase I

Tabla 11.1: Vídeos de la Fase I.

| **FASE I** |
|---|
| **Vídeo 1.1: Seguimiento de pelota en pantalla - OpenCV** |
| En este vídeo se muestra el funcionamiento de un programa que realiza el seguimiento de una pelota utilizando OpenCV.<br><br>En las imágenes capturadas por una cámara se marca la zona en la que se detecta la pelota. También se dibuja una línea que representa el histórico temporal de los últimos movimientos. |
| **Vídeo 1.2: Seguimiento de pelota utilizando un vehículo** |
| En este vídeo se muestra el funcionamiento del programa desarrollado para que un vehículo cambie su dirección siguiendo una pelota.<br><br>En una primera parte de este vídeo se puede observar la detección desde la cámara y cómo la dirección del vehículo acompaña el movimiento de la pelota. En una segunda parte, se establece una velocidad constante y se le permite circular libremente. Colocando la pelota delante del vehículo se puede observar como este avanza hacia la dirección en la que se detecta el objeto. |

## 11.2. Vídeos de la Fase II

Tabla 11.2: Vídeos de la Fase II.

| **FASE II** |
|---|
| **Vídeo 2.1: Resultados del experimento 0041** |
| Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe eliminar árboles y piedras, reconstruyendo en la salida las líneas, el suelo y el cielo.<br><br>Correspondiente al modelo: 0041-simulador-CCCFDDRCtCtCt-0001. |
| **Vídeo 2.2: Resultados del experimento 0042** |
| Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe eliminar árboles y piedras, reconstruyendo en la salida las líneas de la carretera únicamente.<br><br>A diferencia del anterior experimento, en este se espera una salida solo con las líneas.<br><br>Correspondiente al modelo: 0042-simulador-CCCFDDRCtCtCt-0002. |
| **Vídeo 2.3: Resultados del experimento 0043** |

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

A diferencia del anterior experimento, en este se añade una capa convolucional.

Correspondiente al modelo: 0043-simulador-CCCCFDDRCtCtCtCt-0003.

**Vídeo 2.4: Resultados del experimento 0044**

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

A diferencia del anterior experimento, para este se aumenta el vector latente.

Correspondiente al modelo: 0044-simulador-CCCCFDDRCtCtCtCt-0004.

**Vídeo 2.5: Prueba de iluminación – Modelo del experimento 0044**

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

En este experimento se introducen imágenes con diferente nivel de brillo para ver qué tal se comporta un modelo sin haber entrenado con esos casos.

Correspondiente al modelo: 0044-simulador-CCCCFDDRCtCtCtCt-0004.

**Vídeo 2.6: Prueba de iluminación – Modelo del experimento 0045**

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

A diferencia de los anteriores, en este experimento se ha entrenado un modelo con imágenes con diferentes niveles de iluminación. A diferencia de la prueba de iluminación sobre el modelo 0044 (Vídeo 2.5), este modelo sí ha sido entrenado con esos casos y por tanto se reconstruyen correctamente.

Correspondiente al modelo: 0045-simuladorBrillos-CCCCFDDRCtCtCtCt-0001.

**Vídeo 2.7: Resultados del experimento 0046**

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

A diferencia de los anteriores, en este experimento las imágenes de entrada son las líneas capturadas en el entorno simulado superpuestas en paisajes aleatorios.

En el vídeo solo se muestran *frames* de test. Las imágenes de paisajes no se han visto en el entrenamiento en ningún caso.

Correspondiente al modelo: 0046multiGPU-simuladorPaisajes-CCCCFDDRCtCtCtCt-0001.

**Vídeo 2.8: Resultados del experimento 0047**

Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.

Funciona exactamente igual que el experimento anterior (0046) pero en este caso se ha aumentado la variabilidad de las imágenes de la carretera.

Correspondiente al modelo: 0047multiGPU-simuladorPaisajes-CCCCFDDRCtCtCtCt-0002.

| Vídeo 2.9: Resultados del experimento 0052 |
| --- |
| Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe reconstruir en la salida únicamente las líneas de la carretera.<br><br>En este caso se aplica un programa de aumentación de conjuntos de datos precisamente para conducción autónoma. Esta librería se encuentra en GitHub y tiene el nombre de *Automold*. Las modificaciones aplicadas a las imágenes son variaciones de brillo, adición de sombras, nieve, lluvia, niebla, gravilla, reflejos del sol, efecto de velocidad… etc. Además, se aplica ruido sobre la imagen final para complicar aún más la tarea.<br><br>Correspondiente al modelo: 0052multiGPU-simuladorPaisajesAugMIT2-CCCCFDDRCtCtCtCt-0001. |

## 11.3. Vídeos de la Fase III

Tabla 11.3: Vídeos de la Fase III.

| FASE III |
| --- |
| **Vídeo 3.1: Comparativa de radios de giro** |
| En este vídeo se muestran dos configuraciones de las señales PWM del servomotor de dirección.<br><br>En el primer caso se alcanzan radios de giro de 41 cm. Se establecen los valores 400 y 500 como valores PWM límite de giro a izquierda y derecha respectivamente. Esto implica poca maniobrabilidad y construir circuitos muy grandes.<br><br>En el segundo caso se alcanzan radios de giro de 21 cm. Se establecen los valores 350 y 550 como valores PWM límite de giro a izquierda y derecha respectivamente. Con esta configuración se tiene mejor maniobrabilidad. |
| **Vídeo 3.2: Conducción con mando** |
| En este vídeo se muestra el control del vehículo utilizando el mando de la XBOX ONE. |
| **Vídeo 3.3: Determinación de FPS necesarios para conducir - Control con Logitech G29** |
| En este vídeo se conduce a diferente cantidad de FPS para determinar la cantidad mínima necesaria para conducir adecuadamente.<br><br>El simulador utilizado es el City Car Driving. El control se ha realizado con el volante Logitech G29. |
| **Vídeo 3.4: Comparativa de campos de visión** |
| En este vídeo se muestra una comparativa entre dos cámaras.<br><br>Las dos son Pi Camera pero tienen diferentes lentes. La primera tiene una lente normal, con un campo de visión de 62º. La segunda lente es gran angular y tiene un campo de visión de 160º. |
| **Vídeo 3.5: Sistema de control y obtención de datos de entrenamiento** |
| En este vídeo se muestra el sistema de control y obtención de datos de entrenamiento utilizado. |

| |
|---|
| El control utilizado es el volante G29 de Logitech. El servicio de vídeo en tiempo real utilizado es MJPG-Streamer. |

## 11.4. Vídeos de la Fase IV

Tabla 11.4: Vídeos de la Fase IV.

| **FASE IV** |
|---|
| **Vídeo 4.1: Resultado del experimento 0053 - Prueba inicial** |
| Este vídeo muestra un experimento básico. La entrada principal es la imagen capturada en el escenario chroma. La salida corresponde con la misma imagen pero procesada para que solo tenga las líneas de la carretera.<br><br>Nombre del experimento completo: 0053multigpu-escenarioP1-CCCCFDDRCtCtCtCt-0001 |
| **Vídeo 4.2: Secuencia de test con imágenes aumentadas - Modelo del experimento 0057** |
| En este vídeo se muestra una prueba que utiliza el modelo generado en el experimento 0057. Se utiliza una secuencia de test haciendo uso de imágenes aumentadas, es decir, líneas capturadas en el escenario chroma y superpuestas sobre imágenes del escenario objetivo.<br><br>Se superponen las líneas del escenario en imágenes del escenario objetivo (sótano) y se pasan por la red para extraer las líneas de la carretera.<br><br>El modelo utilizado es el generado en el experimento 0057multigpu-escenarioP4-CCCCFDDRCtCtCtCt-0001. |
| **Vídeo 4.3: Secuencia de test en el entorno sótano - Modelo del experimento 0057** |
| En esta prueba se utiliza el modelo generado en el experimento 0057. Se utiliza una secuencia de test en el sótano, es decir, todo lo capturado es real y no hay elementos superpuestos.<br><br>El modelo utilizado es el generado en el experimento 0057multigpu-escenarioP4-CCCCFDDRCtCtCtCt-0001. |
| **Vídeo 4.4: Mejoras incrementales introduciendo ruido - Resultados 0060, 0061, 0062** |
| En este vídeo se pueden visualizar las mejoras incrementales obtenidas en experimentos que utilizan distintos niveles de ruido.<br><br>La secuencia utilizada corresponde con un conjunto de frames obtenidos en el sótano (escenario objetivo). En estos frames todos los elementos son reales, es decir, no hay elementos superpuestos. |
| **Vídeo 4.5: Comparativa entre los experimentos 0057 y 0062 - Influencia del ruido** |
| En este vídeo se puede observar la diferencia entre los resultados de un modelo que no utiliza ruido con respecto a otro que sí lo utiliza. |

| |
|---|
| La secuencia utilizada corresponde con un conjunto de *frames* obtenidos en el sótano (escenario objetivo). En estos *frames* todos los elementos son reales, es decir, no hay elementos superpuestos. |

## 11.5. Vídeos de la Fase V

Tabla 11.5: Vídeos de la Fase V.

| **FASE V** |
|---|
| **Vídeo 5.1: Fases de entrenamiento** |
| En este vídeo se muestra los pasos de las fases de entrenamiento necesarias para obtener el agente de conducción autónoma.<br><br>El sistema se divide en dos fases:<br><br>1. Extracción de características de las líneas de la carretera.<br>- Paso 1: Obtención de imágenes de conducción en el escenario *chroma*.<br>- Paso 2: Obtención de imágenes en el escenario objetivo.<br>- Paso 3: Generación del conjunto de datos de entrenamiento.<br>- Paso 4: Entrenamiento del *Autoencoder*.<br><br>2. Aprendizaje de la conducción.<br>- Paso 1: Obtención de secuencias de conducción en el sótano. Imagen y etiqueta de dirección.<br>- Paso 2: Entrenamiento de la red de decisión.<br>- Paso 3: Puesta en marcha del agente autónomo. |
| **Vídeo 5.2: Secuencia de test con imágenes aumentadas - Experimento 0063** |
| Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe eliminarlo todo, reconstruyendo en la salida las líneas de la carretera únicamente.<br><br>Este experimento prueba la red de extracción de líneas generada en la fase final.<br><br>Los frames pertenecen a una secuencia de test generada con imágenes del escenario chroma y el escenario objetivo. Las imágenes son aumentadas, generadas a partir de dos conjuntos de imágenes. Se sustituyen los colores de suelo y horizonte por imágenes del escenario objetivo, sótano. Estas imágenes sirven para entrenar el autoencoder en ese escenario y probarlo directamente en él a posteriori.<br><br>El nombre completo del experimento en el que se obtiene el modelo utilizado es: 0063multigpu-escenarioVerde2P1-CCCCFDDRCtCtCtCt-0001. |
| **Vídeo 5.3: Secuencia de test en sótano - Experimento 0063** |
| Este vídeo muestra los *frames* de entrada y la salida obtenida por la red. El *Autoencoder* debe eliminarlo todo, reconstruyendo en la salida las líneas de la carretera únicamente.<br><br>Este experimento prueba la red de extracción de líneas generada en la fase final.<br><br>Los *frames* pertenecen a una secuencia de test grabada en el escenario objetivo. Las imágenes de entrada son totalmente reales, es decir, no hay nada superpuesto (el circuito se encuentra en el entorno objetivo). |

| |
|---|
| El nombre completo del experimento en el que se obtiene el modelo utilizado es: 0063multigpu-escenarioVerde2P1-CCCCFDDRCtCtCtCt-0001. |
| **Vídeo 5.4: Conducción autónoma en circuito - Ambos sentidos** |
| En este vídeo se muestra detalladamente una secuencia de conducción autónoma de los agentes obtenidos en los experimentos 0068 (horario) y 0071 (antihorario).<br><br>En primer lugar y principalmente, se muestra en el sentido horario del circuito. Finalmente se muestra una secuencia de conducción autónoma en el sentido contrario (antihorario). |
| **Vídeo 5.5: Conducción autónoma - Motores con mayor velocidad** |
| En este vídeo se muestra una secuencia de conducción autónoma del agente obtenido en el experimento 0068.<br><br>En este caso se utilizan los motores con reductora 1:90, alcanzando una velocidad superior, aunque limitada por el tiempo de inferencia de la Raspberry.<br><br>Se muestra una comparativa entre la velocidad de los motores con reductora 1:90 y 1:120. |

# A. ANEXO I: SEÑALES PWM

En este anexo se describirá el funcionamiento de las señales PWM y, en concreto, cómo se utilizan en el vehículo del proyecto.

## A.1. Funcionamiento de las señales PWM

La necesidad de utilizar señales PWM aparece cuando no es suficiente utilizar una señal digital binaria (apagado o encendido) para enviar a un dispositivo, es decir, es necesario que la señal tenga una intensidad. Por ejemplo, un LED encendido a la mitad de su brillo total, o un motor funcionando a diferentes intensidades para dar diferentes velocidades (uno de los casos que interesa).

Las señales PWM, de las siglas *Pulse Width Modulation* (modulación de ancho de pulsos), sirven para simular una señal analógica a partir de una señal digital.

Estas señales se basan en dos componentes, la frecuencia y el ciclo de trabajo. El ciclo de trabajo (o *Duty Cycle*) indica el porcentaje de tiempo que la señal se encuentra en estado activado del total de la duración de un ciclo. La frecuencia es la que indica la duración del ciclo, indicado en tiempo sería el periodo (el inverso de la frecuencia). Se pueden ver algunos ejemplos de estas señales en la Figura A.1.

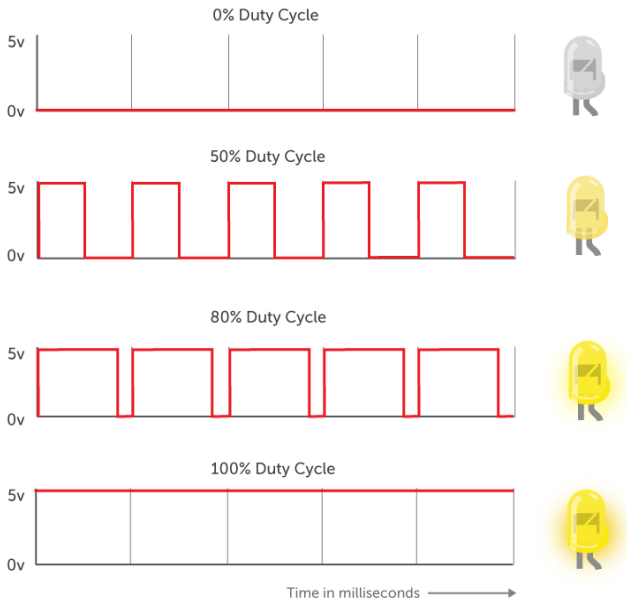



Figura A.1: Ejemplo de funcionamiento de señales PWM.

*En esta figura se muestran ejemplos de funcionamiento del ciclo de trabajo de las señales PWM en un LED. Fuente: [1]*

La señal se comporta como una señal analógica, aunque para conseguir el resultado deseado lo que hace es apagarse y encenderse muy rápido. Es importante que la frecuencia sea suficientemente alta para que el ojo humano no detecte ese apagado y encendido. La tensión final para el dispositivo electrónico será un promedio entre 0-5V dependiendo del ciclo de trabajo. Por ejemplo, con un ciclo de trabajo del 60%, el voltaje del dispositivo será de $0{,}6 \times 5 = 3\,V$.

## A.2. Señales PWM en el vehículo utilizado

En concreto, para utilizar la programación con señales PWM, se estudia el archivo *motor.py* (funcionamiento de los motores de empuje) y *car_dir.py* (funcionamiento del

servomotor de dirección). Estos ficheros se encuentran en el repositorio del código del vehículo [2].

En primer lugar, se definen variables con los pines que se van a utilizar. Los siguientes pines serán utilizados para establecer el sentido de rotación de los motores, es decir, para mover el coche hacia delante o hacia atrás:

```
Motor0_A = 11  # pin11
Motor0_B = 12  # pin12
Motor1_A = 13  # pin13
Motor1_B = 15  # pin15
```

Y los siguientes, son los que se encargan de las señales PWM para cada motor:

```
EN_M0    = 4  # servo driver IC CH4
EN_M1    = 5  # servo driver IC CH5
```

La función que establece la velocidad simplemente escribe en los pines un valor proporcional a la velocidad recibida por parámetro:

```
def setSpeed(speed):
  speed *= 40
  print 'speed is: ', speed
  pwm.write(EN_M0, 0, speed)
  pwm.write(EN_M1, 0, speed)
```

La velocidad viene determinada por una escala de 0-100 de la interfaz de control. Sin embargo, el controlador PWM es de 12 bit y por tanto se le puede escribir desde 0 hasta 4.095 (4.096 pasos). Por lo que la opción más fácil para convertir de 0-100 a 0-4095, es multiplicar por 40 el valor de la entrada (teniendo como máximo 4.000).

También se utilizan señales PWM para controlar el giro del vehículo. El servomotor recibe un valor de intensidad que le indica la posición que tiene que tomar. En el código del coche vienen indicados dos límites establecidos por valores PWM, *leftPWM* y *rightPWM*, estos valores indican la posición del servomotor para girar las ruedas al lado izquierdo y al lado derecho respectivamente. Los valores que se encuentren en el intervalo [*leftPWM*, *rightPWM*] serán valores de giro intermedios.

```
leftPWM = 375
homePWM = 450
rightPWM = 525
```

Teniendo definidos los valores anteriores (home es el valor intermedio, por lo que es tener la dirección recta), se tendrían las funciones necesarias para realizar el giro:

```
def turn_left():
    global leftPWM
    pwm.write(0, 0, leftPWM)

def home():
    global homePWM
    pwm.write(0, 0, homePWM)

def turn_right():
    global rightPWM
    pwm.write(0, 0, rightPWM)
```

Las funciones mostradas escribirían el valor de PWM para girar a izquierda, ir recto o girar a derecha. Sin embargo, estas funciones no permiten realizar giros intermedios, por lo que se utiliza la siguiente función:

```
def turnFromJoystick(value):
    giro = translate(value, -1, 1, leftPWM, rightPWM)
    pwm.write(0, 0, giro)
```

En la función anterior se recibe un valor de giro entre -1 y 1 y se traduce al rango PWM del coche. Con la traducción realizada ya se puede escribir dicho valor y la dirección tomará el giro indicado.

## REFERENCIAS


**[1]** "Raspberry Pi y el PWM | Tienda y Tutoriales Arduino", Prometec.net, 2018. [En línea]. Disponible en: https://www.prometec.net/raspberry-y-pwm/. [Accedido: 02 de mayo de 2020].

**[2]** "Sunfounder_Smart_Video_Car_Kit_for_RaspberryPi", GitHub, 2020. [En línea]. Disponible en: https://github.com/sunfounder/Sunfounder_Smart_Video_Car_Kit_for_RaspberryPi. [Accedido: 02 de mayo de 2020].

# B. ANEXO II: PROCESAMIENTO DE IMAGEN

En este anexo se trata con cuestiones técnicas relacionadas con el procesamiento de imagen. En concreto, con la instalación de OpenCV 4 en una Raspberry y el funcionamiento de un programa para seguir objetos con técnicas de visión artificial.

## B.1. Instalación de OpenCV 4 en Raspbian-Raspberry Pi

El contenido de esta primera parte del anexo se basa en un tutorial [1] publicado por *Adrian Rosebrock.* En este apartado se muestra paso por paso el proceso de instalación de OpenCV 4 en Raspberry.

### B.1.1. Paso 1: Expansión del sistema de ficheros en la microSD

Para realizar este paso es necesario acceder a la configuración de la Raspberry:

```
$ sudo raspi-config
```

Seleccionar *Advanced Options,* y pulsar *Enter* sobre la opción *Expand Filesystem.* Después, salir de la configuración y en caso de que la Raspberry no se reinicie sola, habría que hacerlo manualmente para que se establezcan los cambios correctamente:

```
$ sudo reboot
```

Después del reinicio, el sistema de ficheros se debería haber expandido y lo podemos verificar escribiendo *df -h* en la línea de comandos. Se recomienda tener, al menos, unos 4GB libres en el sistema. Por lo que, si se está utilizando una tarjeta de 8GB, se recomienda eliminar algunos programas como *LibreOffice* o *Wolfram* para liberar espacio:

```
$ sudo apt-get purge wolfram-engine
$ sudo apt-get purge libreoffice*
$ sudo apt-get clean
$ sudo apt-get autoremove
```

### B.1.2. Paso 2: Instalación de dependencias en la Raspberry Pi

Para el correcto funcionamiento de la librería es necesario instalar algunas dependencias. Para ello, es necesario ejecutar los siguientes comandos:

- Actualización del sistema:

```
$ sudo apt-get update && sudo apt-get upgrade
```

- Instalación de algunas herramientas de desarrollador:

```
$ sudo apt-get install build-essential cmake unzip pkg-config
```

- Instalación de algunas librerías de vídeo e imagen que serán esenciales para trabajar con archivos de vídeo e imagen:

```
$ sudo apt-get install libjpeg-dev libpng-dev libtiff-dev
$ sudo apt-get install libavcodec-dev libavformat-dev libswscale-
dev                              libv4l-dev
$ sudo apt-get install libxvidcore-dev libx264-dev
```

- Instalación de GTK, una biblioteca de componentes gráficos para interfaces de usuario:

```
$ sudo apt-get install libgtk-3-dev
```

```
$ sudo apt-get install libcanberra-gtk*
```

- Y, por último, la instalación de *Header files* y herramientas de desarrollo para *Python 3*:

```
$ sudo apt-get install python3-dev
```

### B.1.3. Paso 3: Descarga de OpenCV

Ahora hay que descargar en nuestra carpeta *home* los paquetes de *opencv* y *opencv_contrib* (este paquete contiene módulos y funciones extra):

```
$ cd ~
$ wget -O opencv.zip
https://github.com/opencv/opencv/archive/4.0.0.zip
$ wget -O opencv_contrib.zip
https://github.com/opencv/opencv_contrib/archive/4.0.0.zip
```

Descomprimir lo ficheros descargados:

```
$ unzip opencv.zip
$ unzip opencv_contrib.zip
```

Y, por último, se renombran los directorios para trabajar más fácilmente:

```
$ mv opencv-4.0.0 opencv
$ mv opencv_contrib-4.0.0 opencv_contrib
```

### B.1.4. Paso 4: Configuración de un entorno virtual

En caso de no estar instalado, es necesario instalar *pip*, una herramienta de Python para gestionar paquetes:

```
$ wget https://bootstrap.pypa.io/get-pip.py
$ sudo python3 get-pip.py
```

Ahora, hay que instalar lo necesario para crear entornos virtuales. Creando estos entornos, será posible tener diferentes versiones de software y poder utilizar aquel que convenga en cada momento.

```
$ sudo pip install virtualenv virtualenvwrapper
$ sudo rm -rf ~/get-pip.py ~/.cache/pip
```

Para terminar con la instalación de estas herramientas, es necesario actualizar el archivo `~/.profile`. Este fichero se inicia automáticamente al inicio en el sistema, y se encargará de inicializar lo necesario para trabajar con entornos virtuales. Para modificar el archivo, se puede abrir con cualquier editor de texto como *vi* / *vim* o *nano*, por ejemplo:

```
$ sudo nano ~/.profile
```

Con el fichero abierto y listo para editar, hay que añadir las siguientes líneas al final:

```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

Por último, sería necesario ejecutar el fichero:

```
$ source ~/.profile
```

Teniendo las herramientas instaladas, ya es posible crear el entorno virtual:

```
$ mkvirtualenv cv -p python3
```

Ahora existe un entorno virtual llamado *cv* y se puede acceder a él mediante el siguiente comando:

```
$ workon cv
```

Sabremos que estamos trabajando en el entorno cuando el *prompt* aparezca con el nombre del entorno *(cv)* a la izquierda de una nueva línea en la consola.

Antes de pasar a instalar *OpenCV*, es necesario instalar *numpy*:

```
$ pip install numpy
```

### B.1.5. Paso 5: CMake y compilación en Raspberry

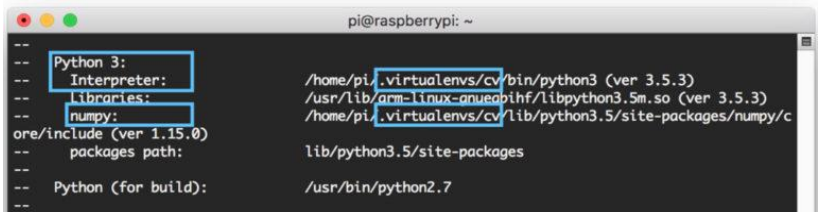
Ahora llega la fase para la que se requiere más tiempo de espera, la compilación. En primer lugar, es necesario crear un directorio para ello:

```
$ cd ~/opencv
$ mkdir build
$ cd build
```

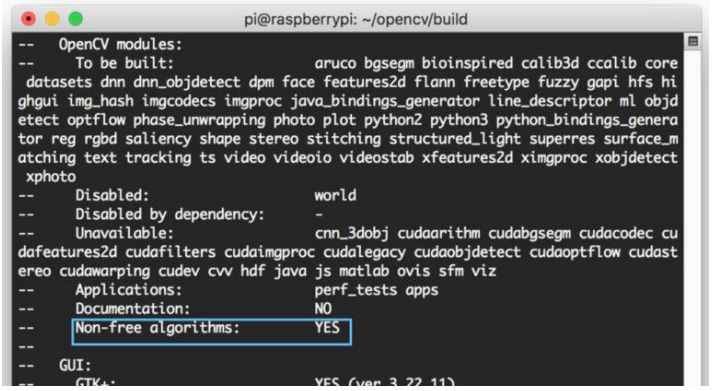
Ahora en el interior de ese directorio, es necesario configurar *CMake* para la instalación de OpenCV 4:

```
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
    -D CMAKE_INSTALL_PREFIX=/usr/local \
    -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
    -D ENABLE_NEON=ON \
    -D ENABLE_VFPV3=ON \
    -D BUILD_TESTS=OFF \
    -D OPENCV_ENABLE_NONFREE=ON \
    -D INSTALL_PYTHON_EXAMPLES=OFF \
    -D BUILD_EXAMPLES=OFF ..
```

Cuando termina este proceso, es importante fijarse en la salida. Se debe mostrar lo que aparece marcado en la Figura B.1.




(a) Directorio de instalación de Python 3 y Numpy




(b) Non-free algorithms

Figura B.1: Capturas para verificar la instalación. Fuente: [1]

Es importante fijarse en que el intérprete de Python y el paquete de *numpy* apunten a la dirección correspondiente al entorno virtual que hemos creado.

Es posible reducir el tiempo de compilación de OpenCV utilizando todo el procesamiento posible de la Raspberry Pi. Para ello, es necesario ampliar el espacio de swap para poder utilizar los cuatro núcleos. Para ello, es necesario abrir el archivo `/etc/dphys-swapfile` y cambiar el tamaño de Swap:

```
$ sudo nano /etc/dphys-swapfile
```

Cambiando el swap de 100 MB (por defecto) a 2.048 MB:

```
CONF_SWAPSIZE=2048
```

Para terminar con el Swap, es necesario reiniciar el servicio:

```
$ sudo /etc/init.d/dphys-swapfile stop
$ sudo /etc/init.d/dphys-swapfile start
```

Ahora estamos listos para pasar a la compilación:

```
$ make -j4
```

Con *-j4* se realiza la compilación usando los 4 núcleos. Sin embargo, haciendo la instalación de esta forma es más posible encontrarse con errores en la instalación. En caso de aparecer algún problema durante la instalación, bastaría con hacer omitir *-j4* para realizar una instalación más lenta pero más segura.

El tiempo total de instalación puede variar en función de la versión de Raspberry Pi que se esté utilizando, capacidad de disipación de calor de la placa o de la velocidad de la tarjeta en la que se realiza la instalación. Aunque en algunos casos pueda parecer que se ha quedado congelado, es algo normal en esta instalación. Un ejemplo de instalación personal: Usando una Raspberry Pi 3B+, una tarjeta microSD de clase 10 y unos pequeños disipadores, el tiempo de instalación ha sido de entre 4 y 5 horas.

Para terminar con la instalación, es necesario ejecutar los siguientes comandos:

```
$ sudo make install
$ sudo ldconfig
```

También es importante dejar el servicio de Swap tal y como estaba:

```
$ sudo nano /etc/dphys-swapfile
      CONF_SWAPSIZE=100
```

### B.1.6. Paso 6: Enlazar OpenCV con Python3

Este paso es necesario para poder importar OpenCV en nuestros programas:

```
$ cd ~/.virtualenvs/cv/lib/python3.5/site-packages/
$ ln -s /usr/local/python/cv2/python-3.5/cv2.cpython-35m-arm-
linux-gnueabihf.so cv2.so
$ cd ~
```

Se recomienda escribir a mano los comandos anteriores intentando realizar completado automático con tabuladores para no confundir las direcciones.

### B.1.7. Paso 7: Comprobar la instalación de OpenCV

Lo primero que hay que hacer es entrar en el entorno virtual creado (es posible que al reabrir la terminal sea necesario cargar de nuevo el archivo *~/.profile* ejecutando *source ~/.profile*. Estando en el entorno virtual hay que lanzar *Python e* importar *OpenCV* para verificar la instalación. Si aparece lo siguiente u otra versión, ya estaría totalmente instalado.

```
$ workon cv
$ python
>>> import cv2
>>> cv2.__version__
'4.0.0'
>>> exit()
```

## B.2. Seguimiento de objetos mediante OpenCV

El contenido de esta segunda parte del anexo se basa en un tutorial [2] publicado por *Adrian Rosebrock* sobre seguimiento del movimiento de objetos mediante *OpenCV*. El contenido se divide en dos partes, la primera dedicada a la explicación de código necesaria para comprender el funcionamiento del seguimiento de objetos. En la segunda parte se explican las modificaciones necesarias para realizar el movimiento de la dirección del coche hacia el objeto detectado.

### B.2.1. Seguimiento de objetos en pantalla

En este apartado se describirá el proceso seguido y el código utilizado para desarrollar el seguimiento en pantalla de una pelota de tenis mediante *OpenCV*. Las explicaciones necesarias se van a realizar haciendo referencia a partes de código incluidas en el documento.

Se parte de la sección de Código B.1: En primer lugar, se hacen las importaciones necesarias (1-8), algunas de ellas son:

- *deque* se va a utilizar como estructura de datos para almacenar los N puntos visitados por la pelota.
- *imutils* será utilizado ya que tiene muchas funciones básicas de procesador de imagen para OpenCV en Python.

Después, algo opcional que se añade al programa es el uso de un *parser* de argumentos (11-16). El primer argumento, *video*, permite dar la dirección de un vídeo en el que hacer el seguimiento de la pelota, en caso de ser omitido, se utiliza la cámara. Y el segundo argumento, sirve para indicar el tamaño del buffer para puntos, es decir, el número de puntos (coordenadas (x,y)) que se guarda como histórico. En caso de omitir este parámetro, será 32 por defecto.

```
1   # import the necessary packages
2   from collections import deque
3   from imutils.video import VideoStream
4   import numpy as np
5   import argparse
6   import cv2
7   import imutils
8   import time
9
10  # construct the argument parse and parse the arguments
11  ap = argparse.ArgumentParser()
12  ap.add_argument("-v", "--video",
13         help="path to the (optional) video file")
14  ap.add_argument("-b", "--buffer", type=int, default=32,
15         help="max buffer size")
16  args = vars(ap.parse_args())
```

Código B.1: Importación de librerías y obtención de la fuente de vídeo

En las líneas 20 y 21 (véase Código B.2), se definen los límites inferior y superior del color que se va a detectar. Esto se hace en el espacio de colores HSV (Matriz (Hue), Saturación (Saturation), Valor (Value)). En la línea 25 se inicializa la variable *pts* como la cola de puntos (estructura de datos) que se van a guardar. Después se inicializan otras variables que serán necesarias, como un contador, posición *x* e *y* y la dirección, que se usará para almacenar y mostrar la dirección actual de la pelota moviéndose en el vídeo.

Para continuar, en caso de que no se haya adjuntado un vídeo como argumento (línea 32, Código B.2), se apunta con la variable *vs* hacía la cámara conectada. En caso de que si se haya introducido un vídeo como argumento (línea 36, Código B.2), se apunta con esa variable al archivo de vídeo. Para hacerlo, se utiliza *VideoStream* de *imutils.video* para manejar los *frames* capturados por la cámara y *cv2.VideoCapture* para el archivo de vídeo. Ahora, se tenga archivo de vídeo o cámara, se puede seguir trabajando de la misma forma.

```
18  # define the lower and upper boundaries of the "green"
19  # ball in the HSV color space
20  greenLower = (29, 86, 6)
21  greenUpper = (64, 255, 255)
22
23  # initialize the list of tracked points, the frame counter,
24  # and the coordinate deltas
25  pts = deque(maxlen=args["buffer"])
26  counter = 0
27  (dX, dY) = (0, 0)
28  direction = ""
29
30  # if a video path was not supplied, grab the reference
31  # to the webcam
32  if not args.get("video", False):
33        vs = VideoStream(src=0).start()
34
35  # otherwise, grab a reference to the video file
36  else:
37        vs = cv2.VideoCapture(args["video"])
38
39  # allow the camera or video file to warm up
40  time.sleep(2.0)
```

Código B.2: Inicialización de variables, preparación de vídeo y definición de límites de color para la máscara

En la línea 43 (Código B.3), comienza un bucle que finalizará cuando se pulse ‘q’ o, en caso de ser un vídeo, cuando no haya más frames (se verán estos dos casos más adelante).

La siguiente instrucción (línea 45, Código B.3) hace una llamada al método *read*(), que lee el siguiente *frame* de cámara o de vídeo. Si se ha introducido vídeo, esta llamada devuelve una tupla de dos elementos, la primera, un *boolean* que indica si el *frame* ha sido leído correctamente, el segundo, es el contenido en sí. En caso de utiliza la cámara, la llamada a *read()* será de *VideoStream*, y se devolverá el *frame* directamente.

En caso de que estemos leyendo un vídeo y el *frame* es *None*, significa que este ha terminado, por lo que termina el bucle (línea 53, Código B.3).

```
42  # keep looping
43  while True:
44        # grab the current frame
45        frame = vs.read()
46
47        # handle the frame from VideoCapture or VideoStream
48        frame = frame[1] if args.get("video", False) else frame
49
50        # if we are viewing a video and we did not grab a frame,
51        # then we have reached the end of the video
52        if frame is None:
53              break
54
55        # resize the frame, blur it, and convert it to the HSV
56        # color space
57        frame = imutils.resize(frame, width=600)
58        blurred = cv2.GaussianBlur(frame, (11, 11), 0)
59        hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
60
61        # construct a mask for the color "green", then perform
62        # a series of dilations and erosions to remove any small
63        # blobs left in the mask
64        mask = cv2.inRange(hsv, greenLower, greenUpper)
65        mask = cv2.erode(mask, None, iterations=2)
66        mask = cv2.dilate(mask, None, iterations=2)
67
68        # find contours in the mask and initialize the current
69        # (x, y) center of the ball
70        cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
71              cv2.CHAIN_APPROX_SIMPLE)
72        cnts = imutils.grab_contours(cnts)
73        center = None
```

Código B.3: Comienzo del bucle principal y procesamiento de la imagen

**Líneas 57-59 (**Código B.3**):** Se inicia el procesamiento de la imagen. En primer lugar, se hace una reducción de tamaño, dejando el ancho a 600 pixeles, así se reduce la cantidad de datos y se podrán procesar datos más rápidamente, teniendo una mayor cantidad de frames procesados por segundo. Después, se aplica un filtro *blur* que servirá para eliminar ruido y enfocar la estructura principal (la pelota verde). Por último, se convierte la imagen al espacio de colores de HSV.

**Líneas 64-66 (**Código B.3**):** Ahora se construye una máscara con la que se detectará la pelota. Esta máscara la vamos a construir con el método *cv2.inRange*, al que le pasamos el *frame*, y el límite inferior y superior de colores HSV. Tras esta operación, lograremos una imagen como la mostrada en Figura B.2:

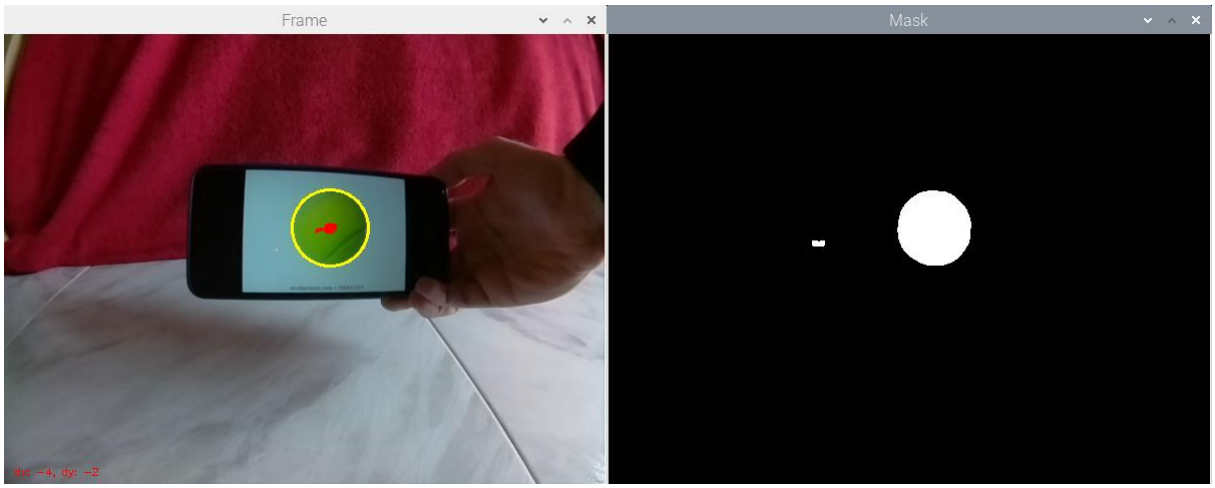



Figura B.2: Aplicación de la máscara para detectar una pelota verde.

*Izquierda: Frame capturado y procesado. Derecha: Máscara utilizada para detectar la pelota.*

Inicialmente en la máscara pueden aparecen manchas o irregularidades, por lo que se aplica un filtro de erosión y dilatación. Estas dos operaciones de *OpenCV*, son operaciones morfológicas y nos permiten eliminar ruido de imágenes. En concreto, erosión permite "adelgazar" el área que compone la imagen (zona blanca) y así eliminar ruido externo a la pelota. El filtro de dilatación hace el efecto contrario, aumentando la zona de la pelota que ha adelgazado. (Con este proceso eliminamos pequeños *blobs*, conjuntos de píxeles)

En la línea 70 (Código B.3) se buscan los contornos en la máscara mostrada. Pasándole los parámetros que se pueden ver, lo que se le está pidiendo a la función es que devuelva el contorno más externo del objeto de la imagen y el más exacto posible, sin usar aproximaciones, respectivamente. En la línea 72, se utiliza *grab_contours* para tomar los contornos del objeto de OpenCV independientemente de la versión de OpenCV.

**Comienzo de Código B.4:** Se continúa con el programa en caso de haber encontrado al menos un contorno. Ahora, ya que puede haber diferentes objetos en la imagen, solo se mantiene el objeto con el contorno más grande (línea 80, Código B.4). A partir del contorno, se obtiene el centro y el radio a partir de la función *minEnconsingCircle*. También se halla el centroide de este contorno, que en OpenCV se halla a partir de *momentos* (línea 83, Código B.4). En caso de que el radio del objeto sea mayor que 10 píxeles, se dibuja un círculo alrededor del objeto usando el radio y el centro obtenido y, además, se actualiza la lista de puntos (líneas 86-92, Código B.4).

```
75          # only proceed if at least one contour was found
76          if len(cnts) > 0:
77                # find the largest contour in the mask, then use
78                # it to compute the minimum enclosing circle and
79                # centroid
80                c = max(cnts, key=cv2.contourArea)
81                ((x, y), radius) = cv2.minEnclosingCircle(c)
82                M = cv2.moments(c)
83                center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
84
85                # only proceed if the radius meets a minimum size
86                if radius > 10:
87                      # draw the circle and centroid on the frame,
88                      # then update the list of tracked points
89                      cv2.circle(frame, (int(x), int(y)), int(radius),
90                            (0, 255, 255), 2)
91                      cv2.circle(frame, center, 5, (0, 0, 255), -1)
92                      pts.appendleft(center)
```

Código B.4: Obtención del contorno más grande encontrado

**Comienzo de Código B.5:** Cuando ya se ha guardado el punto del objeto en el *frame* actual, se recorre la lista de puntos almacenados. En caso de tener 10 o más puntos almacenados, se calcula la dirección que está tomando el movimiento del objeto teniendo en cuenta el punto de un *frame* 10 veces anterior y el actual. Esto se hace así porque tener en cuenta uno tras otro podría dar resultados inestables.

A partir de la línea 111 (Código B.5), en caso de que la pelota se esté moviendo en positivo en X, decimos que la dirección es *West*, y *East* para movimientos negativos (para que sea significativo, debe haber un cambio de 20 pixeles entre los dos *frames* seleccionados). Lo mismo para la *y*, en caso de movernos en positivo significativamente, *North*, en caso contrario, *South.*

```
94          # loop over the set of tracked points
95          for i in np.arange(1, len(pts)):
96                # if either of the tracked points are None, ignore
97                # them
98                if pts[i - 1] is None or pts[i] is None:
99                      continue
100
101               # check to see if enough points have been accumulated in
102               # the buffer
103               if counter >= 10 and i == 1 and pts[-10] is not None:
104                     # compute the difference between the x and y
105                     # coordinates and re-initialize the direction
106                     # text variables
107                     dX = pts[-10][0] - pts[i][0]
108                     dY = pts[-10][1] - pts[i][1]
109                     (dirX, dirY) = ("", "")
110
111                     # ensure there is significant movement in the
112                     # x-direction
113                     if np.abs(dX) > 20:
114                           dirX = "East" if np.sign(dX) == 1 else "West"
115
116                     # ensure there is significant movement in the
117                     # y-direction
118                     if np.abs(dY) > 20:
119                           dirY = "North" if np.sign(dY) == 1 else "South"
120
121                     # handle when both directions are non-empty
122                     if dirX != "" and dirY != "":
123                           direction = "{}-{}".format(dirY, dirX)
124
125                     # otherwise, only one direction is non-empty
126                     else:
127                           direction = dirX if dirX != "" else dirY
```

Código B.5: Calculo de la dirección que está tomando la pelota

**Código B.6:** En este último fragmento de código, se realizan procesos relacionados con lo que se muestra por pantalla. En la línea 131 se calcula el grosor de la línea que sigue la pelota, cuanto mayor sea el número de puntos en el buffer, mayor será el grosor. Por lo demás, se escribe en la imagen la dirección que se está tomando (línea 136) y el diferencial de x e y (línea 138). Por último, se muestra la imagen por pantalla y se aumenta el contador de número de *frames*. En caso de pulsar ‘q’ termina la ejecución, parando el vídeo o liberando la cámara y cerrando la ventana que se había abierto.

```
129               # otherwise, compute the thickness of the line and
130               # draw the connecting lines
131               thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
132               cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)
133
134         # show the movement deltas and the direction of movement on
135         # the frame
136         cv2.putText(frame, direction, (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
137               0.65, (0, 0, 255), 3)
138         cv2.putText(frame, "dx: {}, dy: {}".format(dX, dY),
139               (10, frame.shape[0] - 10), cv2.FONT_HERSHEY_SIMPLEX,
140               0.35, (0, 0, 255), 1)
141
142         # show the frame to our screen and increment the frame counter
143         cv2.imshow("Frame", frame)
144         key = cv2.waitKey(1) & 0xFF
145         counter += 1
146
```

```
147         # if the 'q' key is pressed, stop the loop
148         if key == ord("q"):
149               break
150
151   # if we are not using a video file, stop the camera video stream
152   if not args.get("video", False):
153         vs.stop()
154
155   # otherwise, release the camera
156   else:
157         vs.release()
158
159   # close all windows
160   cv2.destroyAllWindows()
```

Código B.6: Procesamiento de la información que aparece en pantalla para demostrar el seguimiento

### B.2.2. Seguimiento de objetos aplicado al coche

En esta sección se describen las modificaciones añadidas al código anterior para lograr el movimiento de la dirección del coche.

Entre las líneas 50 y 63 (Código B.7), se inicializan los módulos de dirección y motor del coche. Se establece una velocidad constante mediante *motor.setSpeed(value)*, una función que recibe un valor entre 0 y 100 para establecer la velocidad. Se establece el sentido de movimiento mediante *motor.forward()*. Finalmente se inicializa el módulo de dirección del coche con *car_dir.home()*.

```
50  #Setup the busnum
51  busnum = 1
52
53  # Setup steering and motor
54  car_dir.setup(busnum=busnum)
55  motor.setup(busnum=busnum)
56
57  # Constant speed
58  motor.setSpeed(50)
59  motor.forward()
60
61  # Initialize the Raspberry Pi GPIO connected to the servo motor for
    steering.
63  car_dir.home()
```

Código B.7: Inicialización de los módulos del coche

Habiendo inicializado el módulo de dirección, este ya queda preparado para recibir valores de giro y posicionar el servomotor. En el bucle principal, tras detectar el centro del objeto detectado (en las coordenadas (x,y)) se añaden un par de líneas para realizar el giro (Código B.8).

Para realizar el giro tan solo es necesaria la coordenada *x* (horizontal) del centro del objeto. Como se ha explicado en el apartado anterior, el tamaño de la imagen pasa a ser de 600 píxeles horizontales por lo que el rango de la coordenada es [0, 600]. Si la coordenada se encuentra dentro de este rango (línea 128), se hace una llamada a la función de giro que utiliza el coche en este caso, llamada *car_dir.turnFromScreen600(value)* (línea 129).

```
128  if(x >= 0 and x <= 600):
129  car_dir.turnFromScreen600(x)
```

Código B.8: Llamada a la función de giro del coche desde el programa de visión artificial

La función está codificada en el módulo de dirección del coche (Código B.9). La tarea que realiza es pedir la traducción o mapeo del rango dado por el programa de visión artificial ([0, 600]), al rango de giro del coche, establecido en valores PWM. La traducción o mapeo entre rangos se realiza mediante la función especificada en el Código B.10. Tras obtener el giro en el rango que necesita el coche, se escribe el valor de giro sobre el controlador PWM del servomotor y se produce el giro.

```
73  def turnFromScreen600(value):
74      giro = translate(value, 0, 600, leftPWM, rightPWM)
75      pwm.write(0, 0, giro)
```

Código B.9: Función de giro a partir del objeto seguido en el módulo de dirección

La función de traducción o mapeo entre rangos consiste en una serie de operaciones matemáticas que permite traducir de un rango a otro. Para ello se parte del rango actual, el rango deseado y el valor a traducir (Código B.10). Se ha utilizado una entrada de *StackOverflow* [3] para implementar la función de traducción.

```
12  def translate(value, leftMin, leftMax, rightMin, rightMax):
13      #Source: https://stackoverflow.com/questions/1969240/mapping-a-range-of-
    values-to-another
14      # Figure out how 'wide' each range is
15      leftSpan = leftMax - leftMin
16      rightSpan = rightMax - rightMin
17
18      # Convert the left range into a 0-1 range (float)
19      valueScaled = float(value - leftMin) / float(leftSpan)
20
21      # Convert the 0-1 range into a value in the right range.
22      Return rightMin + (valueScaled * rightSpan)
```

Código B.10: Traducción o mapeado entre rangos en el módulo de dirección

## REFERENCIAS


**[1]** A. Rosebrock, "Install OpenCV 4 on your Raspberry Pi – PyImageSearch", PyImageSearch, 2020. [En línea]. Disponible en: https://www.pyimagesearch.com/2018/09/26/install-opencv-4-on-your-raspberry-pi/. [Accedido: 01 de mayo de 2020].

**[2]** A. Rosebrock, "Ball Tracking with OpenCV – PyImageSearch", PyImageSearch, 2020. [En línea]. Disponible en: https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/. [Accedido: 01 de mayo de 2020].

**[3]** M. another, T. Mawushe, A. Luchjenbroers and J. Cann, "Mapping a range of values to another", Stack Overflow, 2020. [En línea]. Disponible en: https://stackoverflow.com/questions/1969240/mapping-a-range-of-values-to-another. [Accedido: 14 de junio de 2020].

# C. ANEXO III: EXPERIMENTACIÓN PREVIA

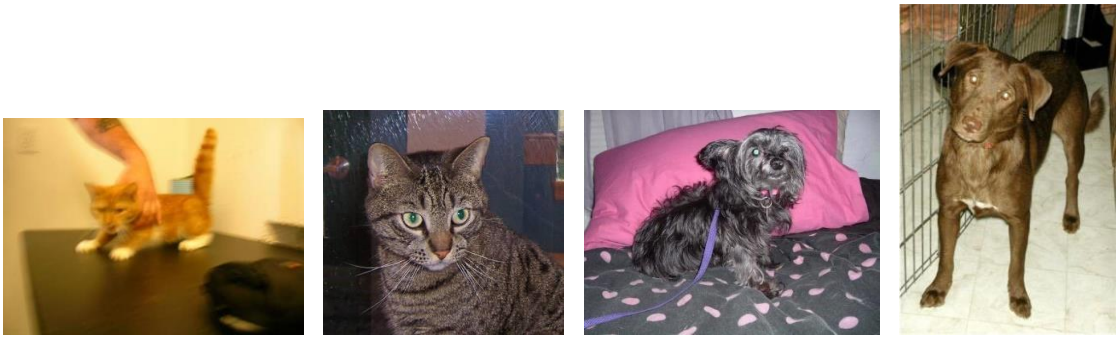
En este anexo se incluye parte de la documentación de experimentación previa realizada durante el proyecto.

## C.1. Experimentación en el dominio de Dogs vs Cats

El dominio de *Dogs* vs *Cats* [1] está compuesto por 25.000 imágenes etiquetadas: 12.500 de perros y 12.500 de gatos. Las imágenes del conjunto son imágenes a color con tamaño variable.

Este ha sido el primer dominio utilizado para realizar experimentación y aprender sobre redes de neuronas. Este dominio es muy común dentro del mundo de la visión artificial y consiste en clasificar imágenes que contienen un perro o un gato. Es considerado por muchos como el "*Hello world*" de las redes de neuronas.

En este apartado se presentarán las conclusiones de la experimentación realizada y se abordará un tema muy interesante de este campo, la visualización de redes de neuronas.



(a) Ej. 1 (b) Ej. 2 (c) Ej. 3 (d) Ej. 4

Figura C.1: Imágenes de ejemplo del conjunto de *Dogs vs Cats*.

Aunque el conjunto de datos sea de 25.000 imágenes, en los experimentos realizados han sido utilizadas únicamente un total de 4.000, 2.000 de perros y 2.000 de gatos. De esta cantidad, se han destinado 2.000 imágenes para entrenar, 1.000 para realizar la validación y otras 1.000 para realizar el test. En todo momento se mantiene una proporción de ejemplos equilibrada entre ambas clases.

La experimentación realizada en este dominio ha sido muy útil para conocer muchas de las técnicas que se pueden utilizar para obtener mejores modelos. Las técnicas que se han utilizado en dichos experimentos han sido muy variadas, las que mejor resultado han ofrecido han sido las siguientes:

- *Data Augmentation*.
- Uso de la regularización Dropout.
- Uso de diferentes optimizadores (RMSProp, Adadelta…)

Además, esta experimentación ha servido para aprender a trabajar utilizando una metodología de experimentación concreta, especificada anteriormente. El concepto más importante aprendido fue la metodología experimental, de manera que sea posible comparar los experimentos y saber qué cambio ha sido el responsable de la mejora.

### C.1.1. Visualización de redes de neuronas

Un aspecto que ofrecen las redes convolucionales a diferencia de otras técnicas de aprendizaje automático es la posibilidad de visualizar las representaciones que aprenden.

Existen muchas técnicas para visualizar estas representaciones, en este apartado serán descritas dos de ellas. Además, se mostrarán ejemplos de visualización extraídos de los experimentos realizados en el dominio de Dogs vs Cats.

**Visualización de activaciones intermedias**

Esta técnica consiste en mostrar los mapas de características que se dan a la salida de las capas convolucionales y de *pooling*. Dada una entrada en una capa convolucional, en la salida de esta se obtiene la activación de los filtros de la operación de convolución. Esta forma de describirlo da una perspectiva de cómo la entrada se descompone en los diferentes filtros aprendidos por la red.

Para visualizar los mapas de características se sigue el siguiente proceso:

En primer lugar, se carga el modelo del que se quiere obtener los mapas de características. Teniendo el modelo cargado, se crea otro que tendrá la misma entrada pero tendrá múltiples salidas. Este nuevo modelo tendrá una salida por cada capa convolucional o de *pooling* que tenga el modelo inicial.

En segundo lugar, teniendo creado el nuevo modelo de múltiple salida, es necesario introducir una entrada en él. La salida dada por el modelo serán las activaciones de las distintas capas y tan solo es necesario mostrarlo para lograr ver los mapas de características.

En los ejemplos que se van a visualizar, han sido representados conjuntamente los filtros (canales) de las distintas capas. La visualización se realiza mediante la librería Matplotlib usando el mapa de color *viridis.* En la Figura C.2 se puede observar cómo ese canal trata de representar bordes.

Figura C.2: Canal treinta de la primera capa convolucional en *Dogs vs Cats*.

En la Figura C.3 se pueden observar algunos otros canales que detectan únicamente líneas o bordes. Sin embargo, en la Figura C.4 se pueden visualizar características más complejas. Por ejemplo, el canal que aparece en la primera fila y cuarta columna aparenta detectar los ojos.

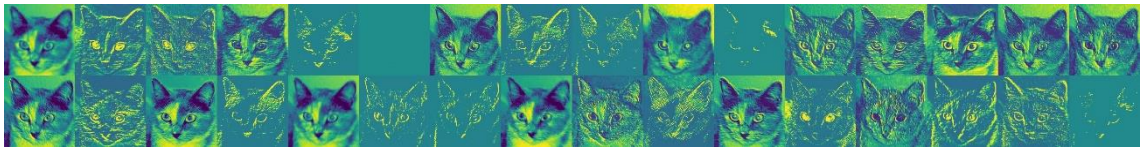


Figura C.3: Filtros de la primera capa convolucional en *Dogs vs Cats*.

*La primera capa convolucional de esta red está compuesta por 32 filtros.*

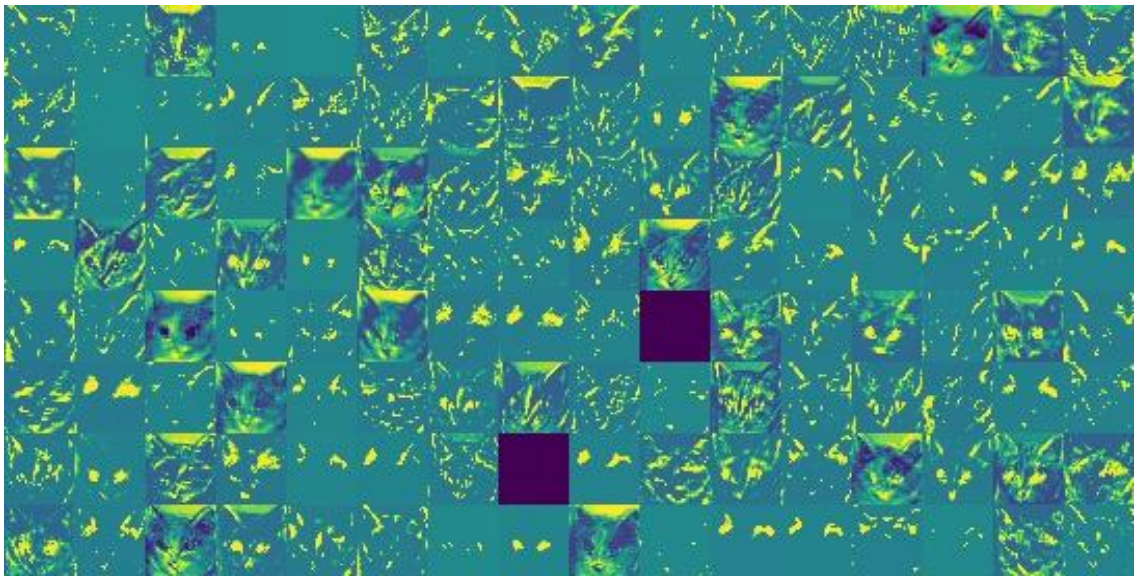


Figura C.4: Filtros de la tercera capa convolucional en *Dogs vs Cats*.

*La tercera capa convolucional de esta red está compuesta por 128 filtros.*

Con este conjunto de mapas de características, una red de neuronas posterior podría aprender que, si se ha activado el filtro que detecta orejas de gatos y el filtro que detecta ojos de gato, en la entrada nos encontramos una imagen de esta clase.

Un detalle que se puede observar en la Figura C.4 es la existencia de dos canales "apagados". Esto se debe a que las activaciones para la entrada introducida en esos canales son nulas ya que el patrón detectado por dicho canal no se ha encontrado en la imagen.

Como conclusión, se ha podido observar experimentalmente como en las primeras capas se detectan bordes, características simples. Sin embargo, cuando se avanza a capas convolucionales más profundas, las características detectadas son más abstractas.

**Visualización de mapas de activación de clase**

Esta técnica de visualización es útil para entender qué partes de una imagen dada a una red convolucional son importantes en una tarea de clasificación. La técnica general recibe el nombre de *Class Activation Map* (CAM). Consiste en generar mapas de calor según la clase activada sobre la imagen de entrada.

La implementación utilizada para obtener los ejemplos que se van a mostrar recibe el nombre de *Grad-CAM*. Se utiliza este método ya que es una forma general de obtener los mapas de calor a partir de los gradientes, sin necesidad de modificar la estructura de la red.

El método pretende visualizar la importancia de cada neurona en la decisión que se toma. Una descripción a grandes rasgos del proceso que se realiza es la siguiente: Dada una entrada, se toma el mapa de características de una capa convolucional. El método consiste en ponderar cada canal de ese mapa de características utilizando el gradiente de la clase con respecto al canal correspondiente.

Una forma intuitiva de verlo es fijarse en lo siguiente: se ponderan los canales de los mapas de características activados sabiendo cómo de importantes son esos canales con respecto a la clase (mediante el gradiente).

Para probar esta técnica de visualización, se ha utilizado como entrada una imagen propia, es decir, no ha participado en el proceso de entrenamiento, validación o test del modelo creado. El modelo utilizado para extraer el mapa de calor ha sido el 0007-CvD-CMCMCMCMFDrDD-0007.

Como se puede observar en la Figura C.5, se marcan las zonas de la imagen que han sido útiles para clasificar el ejemplo dentro de la clase *Perro*. Aunque se marca alguna zona que carece de sentido, la zona más marcada es la cabeza.

Figura C.5: Mapa de calor de un ejemplo para la clase *Perro*.

*Las zonas marcadas con colores cálidos son las que han tenido importancia para realizar la tarea de clasificación.*

Como curiosidad, se han extraído mapas de calor para el mismo ejemplo mostrado anteriormente mediante la red VGG-16 para visualizar como realiza la tarea de clasificación esta red.

La VGG-16 es una red convolucional propuesta por K. Simonyan y A. Zisserman [2]. Esta red consiguió alcanzar un 92,7 % de precisión sobre el conjunto de imágenes de ImageNet [3]. Este conjunto de imágenes consta de unos 15 millones de imágenes que corresponden a unas 22.000 categorías.

En la Figura C.6 se muestra el mapa de calor para el ejemplo anterior y otra imagen usando la red VGG-16. En estos mapas de calor se puede observar como la zona marcada es más precisa, e incluso en la imagen a de dicha figura se observa como se ha marcado una zona característica de esta raza de perros, las orejas. Dado que esta red está entrenada sobre muchos ejemplos y sobre muchas clases, es comprensible que el resultado sobre esta red sea más preciso.

(a) Ej. 1 (b) Ej. 2

Figura C.6: Mapas de calor para dos imágenes sobre la red VGG-16.

*La clase en la que han sido clasificadas ambas imágenes ha sido 'Beagle'.*

## C.2. Experimentación en el dominio de Sign Language MNIST

El conjunto de datos de *Sign Language MNIST* [4], de aquí en adelante *Hand Sign Recognition – Letters* (HSRL), es un conjunto de Kaggle que está compuesto por imágenes del lenguaje de signos para las letras. Este conjunto de datos tiene el objetivo de ser similar que el gran conocido MNIST, al igual que han hecho con otros como, por ejemplo, Fashion-MNIST [5].

El conjunto de datos está formado por imágenes de 28x28 píxeles en escala de grises pertenecientes a 25 clases. No existen casos para la letra J y Z ya que la representación de estas letras requiere de gestos con movimiento. El repositorio ofrece directamente el conjunto de entrenamiento y de test formados, teniendo un total de 27.455 imágenes para entrenamiento y 7.172 imágenes para la fase de test.

Algo importante que tener en cuenta de este conjunto de datos es que proviene de una cantidad pequeña de imágenes. Los creadores del conjunto inicialmente disponían de 1.704 imágenes. Aunque mediante técnicas de aumentación consiguieron aumentar el conjunto hasta alcanzar la cantidad mencionada anteriormente.

Usar este conjunto de datos ha permitido realizar más experimentación para afianzar más los conocimientos adquiridos. Además, con este conjunto se ha practicado también con el pre procesado de datos.


Figura C.7: Clases del conjunto de datos *Sign Language MNIST*.

*Las imágenes del conjunto de datos utilizado son en blanco y negro.*

Inicialmente se desconocía que los conjuntos de datos ofrecidos por el repositorio habían sido aumentados, por lo que se decidió unificar ambos para obtener las particiones de entrenamiento, validación y test al gusto. Al realizar experimentación con estos conjuntos se obtenían tasas de acierto muy altas, incluso superiores en validación que en entrenamiento. Esto llevo a pensar que los conjuntos de entrenamiento y validación tenían ejemplos muy parecidos y que se había cometido algún error.

Después de este inconveniente, se volvieron a tomar los conjuntos dados por el repositorio y se formó el conjunto de validación dividiendo en dos partes el conjunto de test. De esta forma sí fue posible realizar experimentación en el conjunto. A continuación, se describen los experimentos más relevantes.

Los dos primeros experimentos del dominio se realizaron con los conjuntos de datos incorrectamente formados, como se ha indicado antes, por lo que no se muestran. El primer experimento válido de este dominio es el 0018-HSRL-CMCMCMFDD-0003. Este experimento ofrece un acierto del **93,9 %** en el conjunto de test. Aunque inicialmente el resultado ya es bastante bueno, se han realizado variaciones en los experimentos para maximizar este acierto.

En el siguiente experimento, 0019-HSRL-CMCMCMFDrDD-0004, se añade una capa de Dropout tal y como se puede observar en la topología. Al añadir esta regularización, se consigue aumentar el porcentaje de acierto en test al **95,8 %**. Esta regularización dispone de un parámetro que indica la probabilidad con la que las neuronas se “apagan”. En este experimento esa probabilidad se establece al 25 %. En el siguiente experimento se prueba con un 50 % y se obtiene el mismo resultado, por lo que se fija en 25 %.

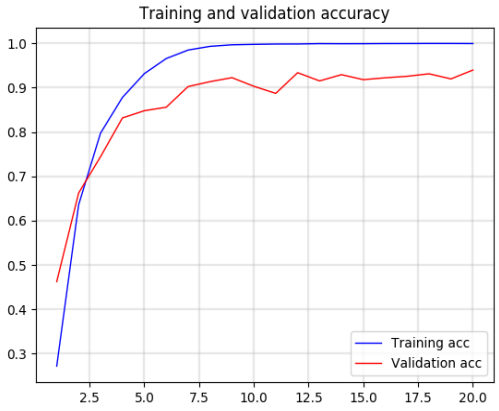



Figura C.8: Experimento 0018-HSRL-CMCMCMFDD-0003.

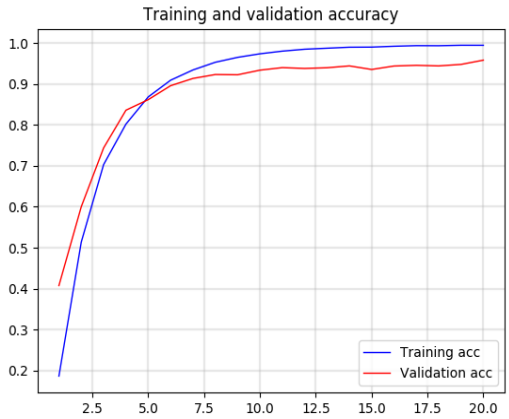



Figura C.9: Experimento 0019-HSRL-CMCMCMFDrDD-0004.

En los siguientes experimentos se decide utilizar d*ata augmentation*. Esta técnica es útil para aumentar conjuntos de datos pequeños o poco variados. Consiste en aplicar transformaciones a las imágenes de forma que se consiga multiplicar la cantidad de casos diferentes. Aplicando esta técnica se consiguen redes más robustas, capaces de generalizar más. En la Figura C.10 pueden observarse ejemplos de la aplicación de esta técnica.

Figura C.10: Aplicación de *data augmentation* sobre el dominio HSRL.

*La primera imagen de cada fila es la original. El resto de las imágenes de cada fila tienen una rotación aleatoria dada por un pequeño rango.*

En el experimento 0023-HSRL-CMCMCMFDrDD-0008 se aplica *data augmentation* sobre el conjunto de entrenamiento aplicando una rotación aleatoria en el rango [-20, 20] grados. Al utilizar esta técnica, el conjunto de entrenamiento es más complejo (tiene más variedad). En este experimento se aumentan los ciclos de entrenamiento lo suficiente como para asegurar la convergencia. Se obtiene un acierto del **96,61 %** en el conjunto de test, mejorando el resultado anterior.

Por último, en el experimento 0024-HSRL-CMCMCMFDrDD-0009 se añade una variación más a las imágenes. Se mantiene el giro anterior y se añaden zooms aleatorios

de un 0,1, es decir, zooms aleatorios entre el 90 % (alejar la imagen) y el 110 % (acercar la imagen). En este experimento se obtiene un acierto del **97,32 %** en el conjunto de test.

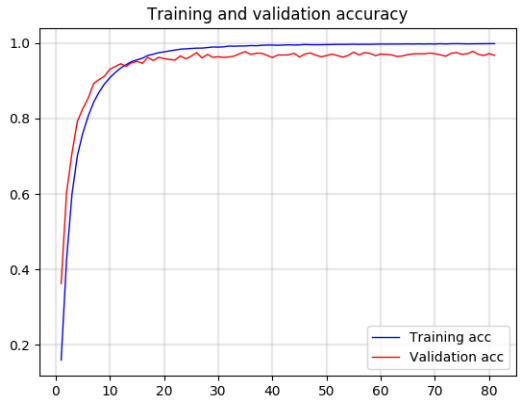



Figura C.11: Experimento 0023-HSRL-CMCMCMFDrDD-0008.

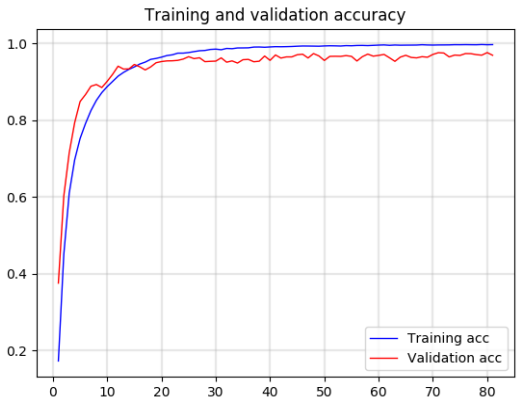



Figura C.12: Experimento 0024-HSRL-CMCMCMFDrDD-0009

Aunque las mejoras son muy pequeñas, se puede observar cómo al aplicar la técnica de *data augmentation* se consigue entrenar modelos capaces de acertar una mayor cantidad de ejemplos.

Al igual que en el dominio de *Dogs vs Cats*, en este también se han utilizado técnicas de visualización de redes de neuronas. Dado que se utilizan imágenes muy pequeñas, en la tercera capa convolucional los mapas de características son de tamaño 7x7, en los que es imposible visualizar características. Sin embargo, en los filtros de las dos primeras capas convolucionales sí puede observarse cómo se obtienen contornos o detección de bordes (véase Figura C.13 y Figura C.14).

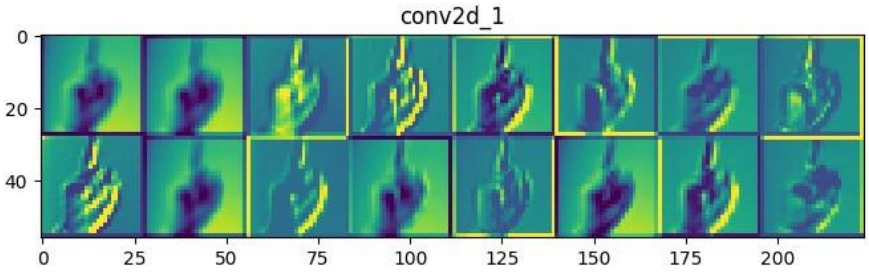



Figura C.13: Filtros de la primera capa convolucional en HSRL.

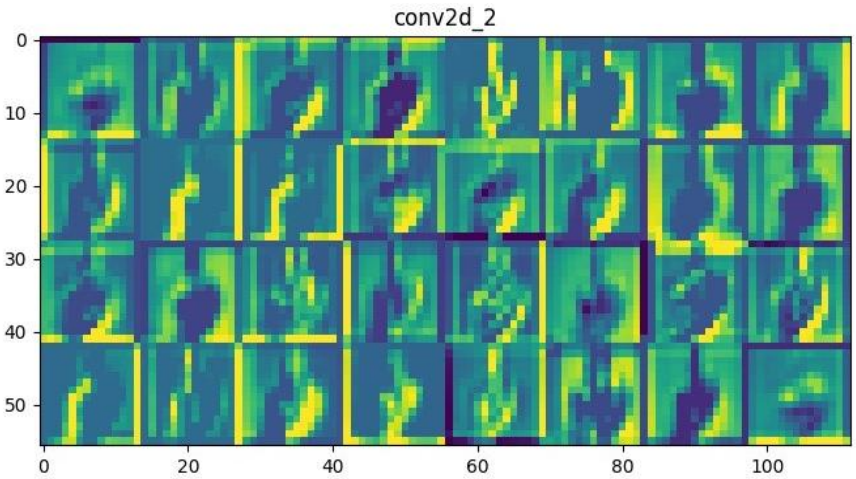



Figura C.14: Filtros de la primera capa convolucional en HSRL.

Aprovechando que nos encontramos ante un problema de clasificación, también se han podido obtener mapas de calor mediante la técnica de *Grad-CAM*. Se incluyen algunos ejemplos en la Figura C.15. Estos mapas de calor han permitido visualizar cómo la red aprende fijándose en la mano.

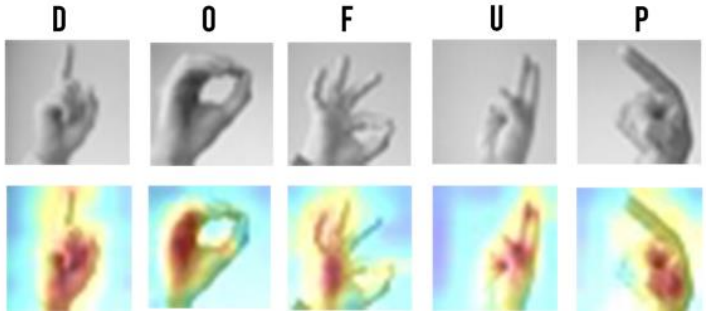



Figura C.15: Mapas de calor en ejemplos de HSLR.

*El color con el que se marcan las zonas indica la importancia para clasificar el ejemplo. Los colores cálidos indican las zonas más importantes.*

Tras ver unos resultados tan positivos, se realizó una prueba con imágenes propias. Sin embargo, los resultados obtenidos en esta prueba no fueron satisfactorios. Esto demostraba que la red es muy dependiente del dominio y la generalización es muy baja.

Una de las principales aportaciones de este dominio a lo aprendido es el problema de generalización de las redes de neuronas. Sería necesario entrenar la red con ejemplos más variados, por ejemplo, diferentes manos, diferentes fondos, diferentes colores… Esto último podría mejorar la generalización de la red, haciendo que esta únicamente atienda a características mostradas por cualquier mano que realiza el gesto y no otros elementos del entorno o una mano en particular.

## REFERENCIAS

**[1]** «Dogs vs. Cats». https://kaggle.com/c/dogs-vs-cats (accedido abr. 29, 2020).

**[2]** K. Simonyan y A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition. 2014.

**[3]** J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, y L. Fei-Fei, «Imagenet: A large-scale hierarchical image database», 2009 IEEE conference on computer vision and pattern recognition, 2009, pp. 248–255.

**[4]** «Sign Language MNIST». https://kaggle.com/datamunge/sign-language-mnist (accedido abr. 29, 2020).

**[5]** «Fashion MNIST». https://kaggle.com/zalando-research/fashionmnist (accedido abr. 29, 2020).

# D. ANEXO IV: MULTI – GPU EN *DEEP LEARNING*

En este anexo se describirán aspectos técnicos sobre el entrenamiento de redes de neuronas haciendo uso de más de una unidad gráfica de procesamiento (GPU).

## D.1. Ordenador de experimentación y Entorno software

En este apartado se muestran los componentes del ordenador utilizado y el entorno software sobre el que se ha realizado la experimentación con *Deep Learning*.

Componentes del ordenador utilizado:

- **Placa base**: Asus Prime Z370-P II
- **Procesador**: Intel Core i5-9400F 2.9GHz 6 núcleos
- **Memoria RAM**: 16 GB DDR4
- **GPU**: 2 x GeForce GTX 1050Ti 4 GB; **Compute capability**: 6.1

Entorno software:

- **Keras**: 2.3.1
- **Tensorflow – GPU**: 1.13.1
- **CUDA**: 10.0.130
- **CUDNN**: 7.6.5
- **NVIDIA GPU Driver**: 440.33.01
- **Sistema Operativo**: Ubuntu 18.04.3 LTS

## D.2. Multi – GPU en Keras

La experimentación se ha realizado utilizado Keras como interfaz de alto nivel y Tensorflow como *backend*. Para realizar el entrenamiento de redes de neuronas haciendo uso de dos GPU se ha utilizado un fichero de código de un repositorio alojado en GitHub [1] y se ha configurado una variable de entorno. A continuación se describen los aspectos mencionados.

### D.2.1. Código utilizado

En el repositorio al que se ha hecho referencia existe un fichero llamado *multi_gpu.py*. Este dispone del código necesario para preparar el modelo a entrenar utilizando múltiples GPU, en este caso se utilizarán las dos tarjetas gráficas ya mencionadas.

Es necesario descargar este fichero e importar la función multi_gpu en el fichero de código en el que estamos creando nuestro modelo:

```
from multi_gpu import to_multi_gpu
```

Teniendo la función ya importada y un modelo de red neuronal definido, se realiza la llamada a la función anterior para configurar la estructura del modelo de forma que sea posible entrenarlo en varias GPU. Los parámetros de esta función son: el modelo que se quiere configurar y la cantidad de tarjetas gráficas que se van a utilizar en el entrenamiento. Esta llamada se realiza antes de compilar el modelo.

```
model = to_multi_gpu(model,n_gpus=2)
```

Después de dicha llamada, sería necesario compilar el modelo estableciendo los parámetros deseados, este paso se realiza de la misma forma que en condiciones de entrenamiento normal (mediante CPU o una sola GPU).

En la Figura D.1 se muestra un ejemplo obtenido del repositorio mencionado. En él se puede observar un ejemplo completo de la definición de una red neuronal, el ajuste para entrenarlo en múltiples GPU y, por último, la compilación del modelo.

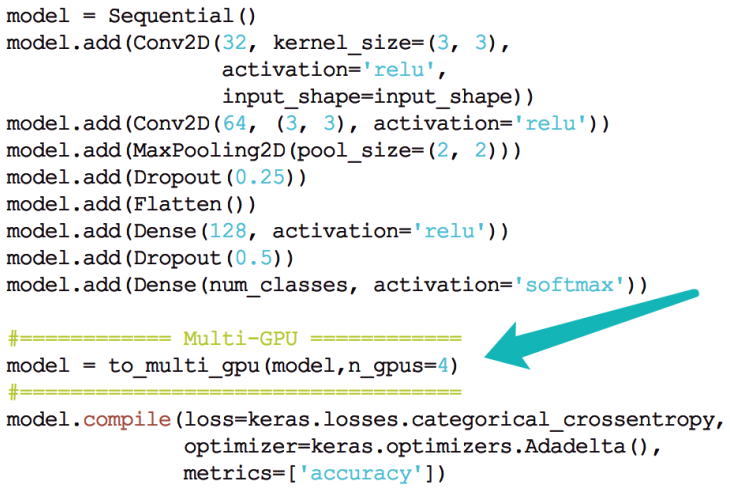



Figura D.1: Ejemplo completo de uso de entrenamiento en multi GPU.

*En la figura se muestra un ejemplo completo de la definición de una red neuronal, la llamada a la función encargada de ajustar el modelo para ser entrenado en varias GPU y, por último, la compilación del modelo creado. En el caso de la imagen se utilizan 4 GPU. Fuente: [1]*

Una vez realizados estos pasos, es necesario configurar el entorno, en el siguiente apartado se describen los pasos a seguir.

### D.2.2. Configuración de variables de entorno

Antes de ejecutar el fichero de código que entrene la red neuronal creada es necesario configurar una variable de entorno.

Esta variable de entorno permite a CUDA conocer las GPU que se pueden utilizar en el proceso de entrenamiento. En este caso, dado que se desean utilizar ambas tarjetas gráficas del ordenador, es necesario indicarlo de la siguiente forma en la terminal:

```
$ export CUDA_VISIBLE_DEVICES=0,1
```

Una vez ejecutado el comando anterior, CUDA podrá utilizar las GPU con identificador 0 y 1, que corresponden con la primera y segunda que hayan sido identificadas. Esto último se puede consultar lanzando el comando `nvidia-smi` que ofrece información sobre las unidades gráficas instaladas en el ordenador

Una vez realizado este paso ya es posible entrenar el modelo creado usando multi GPU.

Además de poder realizar el entrenamiento de un modelo utilizando dos GPU, también es posible entrenar dos modelos al mismo tiempo, uno en cada tarjeta gráfica instalada. Para ello será necesario abrir dos terminales, y en cada una establecer una de las tarjetas gráficas como visible.

Primera terminal → `$ export CUDA_VISIBLE_DEVICES=0`

Segunda terminal → `$ export CUDA_VISIBLE_DEVICES=1`

De esta forma, cuando se ejecute el programa que entrena cada modelo en cada terminal, se utilizarán diferentes GPU para ello.

### D.2.3. Experimentación utilizando Multi – GPU

En este proyecto se han entrenado los modelos más pesados haciendo uso de dos GPU, en concreto, los *Autoencoders* de la tarea de extracción de líneas de la carretera.

A continuación se muestra una comparación del tiempo de entrenamiento para un solo ciclo en uno de los modelos creados, específicamente, el modelo 0046multiGPU-simuladorPaisajes-CCCCFDDRCtCtCtCt-0001. Se utiliza un *Batch Size* de 16 dado que a partir de esa cantidad aparecen errores de control de memoria, por lo que se ha decidido no tener en cuenta dichos tiempos.

1 GPU → 263 segundos por ciclo de entrenamiento.

2 GPU → 178 segundos por ciclo de entrenamiento.

Se puede observar cómo el tiempo de entrenamiento se reduce bastante al utilizar ambas tarjetas gráficas. Teniendo en cuenta que algunos modelos han necesitado entrenar durante 40 ciclos, con una GPU el proceso hubiese terminado en unas 3 horas. Sin embargo, utilizando ambos dispositivos ha sido posible completar el entrenamiento en 2 horas, reduciendo en total una hora por modelo entrenado.

## REFERENCIAS

**[1]** "kuixu/keras_multi_gpu", GitHub, 2020. [En linea]. Disponible en: https://github.com/kuixu/keras_multi_gpu. (accedido jun. 25, 2020).

# Development of a Reduced Scale Car for Autonomous Driving Learning through Deep Learning Techniques

## E. ANEXO V: EXTENDED ABSTRACT

### E.1. INTRODUCTION

Experiments on autonomous driving have been performed since the 1920s. It was in the 1950s that promising results began to appear that caused interest in the subject.

The first vehicles with autonomous driving functions, such as road monitoring, appeared in the 1980s. In 1984, the Carnegie Mellon University developed a project called ALVINN [1] and in 1987 several scientists from European Universities were involved with the EUREKA PROMETHEUS project [2].

Technological advances in autonomous driving are growing at a vertiginous rate and are gradually being integrated into society. The possibility of reducing the number of accidents, reducing the effort required to drive and facilitating travel for the disabled are some of the advantages of autonomous vehicles.

In recent years, companies such as Google [3], General Motors [4] and Tesla [5], among many others, have been involved with the research and implementation of autonomous driving systems. Today, the most advanced commercial autonomous vehicles control direction and speed in non-urban areas. Although there is still a need for the driver to pay attention to driving, technological and regulatory developments will allow full autonomous driving in the future. In fact, tests are already being performed with almost fully autonomous vehicles in some public areas in the USA [6].

The present work will develop a small-scale vehicle that learns to drive autonomously. It must have the necessary components to perform the drive and to deal with the necessary tasks. These will be mainly the traction and steering motors, the power supply, the on-board computer, and a camera.

It will also be necessary to develop a driving circuit that reproduces a road delimited by two lines on which the vehicle can circulate and learn to drive autonomously.

Using an external computer and a control device (a steering wheel) linked to it, a system will be developed that allows vehicle control and obtaining training data sets. This system will communicate the computer and the vehicle wirelessly.

To achieve the development of the agent responsible for driving, an approach divided into two tasks is proposed.

Firstly, a feature extraction of the road lines from the captured image will be performed. In this work, this task is approached using a type of neural network called Autoencoder. Using this technique, a set of features can be obtained in an unsupervised way.

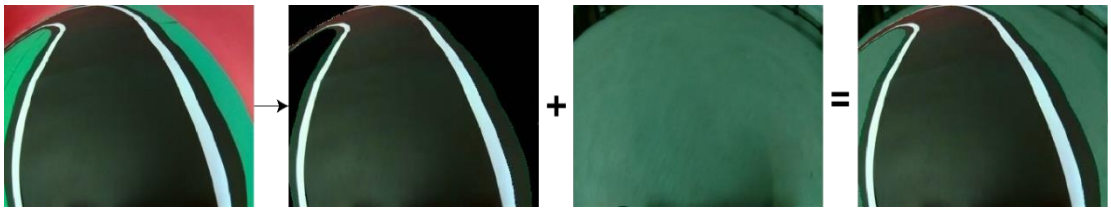
Secondly, the features obtained will be used to carry out the training of driving in a supervised way.

Since supervised learning is used, it will always be essential to have for each situation the necessary labels that clearly identify what action should be taken in each case. This is

solved by the driving environment in which the actions of an operator are recorded while he drives the vehicle.

The data labelling in the task of extracting the lines from a road is more complex. It is necessary to use some technique that allows to differentiate them from the rest of the elements. A manual labelling process is not considered because it is too tedious, and the use of Computer Vision techniques is discarded because of the possible failures it may cause.

For this purpose, we propose the construction of an auxiliary stage (based on chromas) to facilitate the extraction of the lines. These lines can be virtually superimposed on images of the driving scenario. These augmented images will serve to train the Autoencoder. In this way, this network will learn to isolate the lines in real scenarios, making it possible to learn to drive in them.



(a) Image captured on chroma stage

(b) Image processed. Isolated road.

(c) Image captured on target stage

(d) Resulting augmented image

Figure E.1: Flow of the augmented images generation process.

This document follows the following structure: the **Introduction** presents the general idea of the project; the **State of the Art** deals with the knowledge about aspects of interest of autonomous driving and neuronal networks; the **Analysis and Design**, describes the problem and the design used to solve it; the **Experimentation** presents the most relevant experimentation of the project; the **Validation**, describes tests to know the scope of the system; at last, the **Conclusions and Future Work** describes the conclusions of the project and possible future lines of work.

## E.2. STATE OF ART

### E.2.1. Autonomous driving

Nowadays, there are different methods for dealing with autonomous driving systems.

Some systems use techniques to understand the environment and to make decisions based on the information provided by a variety of sensors and devices. This approach is currently the basis of the most remarkable autonomous driving systems.

However, other systems are based only on the use of an image and the application of Computer Vision and Deep Learning techniques. There are two main approaches based on this scheme:

- The first is typically called mediated perception. The input image is used to understand the environment using Computer Vision Techniques that provide an extraction of explicit features. This information allows route planning and control. This is the most common approach.
- The second approach is called behaviour reflex. It consists of mapping the pixels of the input image directly to control commands, without the need to understand the environment explicitly. The objective of the system is to learn how to behave

from training in which the captured images are recorded together with driving commands.

The learning of these systems is usually supervised. Autonomous agents trained in this way are based on Behavioural Cloning. The main idea of this method is to learn how to perform a task from observations and to imitate it.

There is also the possibility of using unsupervised learning. This other technique would allow the agent to develop a driving ability superior to the human one. However, these types of agents are more difficult and slower to train.

### E.2.2. Neural Networks

#### E.2.2.1. Regularization – Introduction of noise

During the learning process, multiple techniques can be applied to improve the obtained network. These techniques are called regularizers and are mainly intended to improve the generalization error of the network.

The addition of noise to the input data modifies the training cases so that it is possible to expand the data set and obtain a more robust network. This will be able to deal with data with noise and will also improve its generalizability ability on data without noise. The effect can also be related to the fact that by training the network with noise, more neurons in the network contribute effectively to its operation [7].

#### E.2.2.2. Autoencoders

An Autoencoder (AE) is a type of neural network whose purpose is to copy the input data to the output. It has a hidden layer that describes a representation of the input, this is called a latent vector. This network is divided into two parts, the encoding part (Encoder) and the decoding part (Decoder).

There are variants and techniques that improve the basic Autoencoder.

Using more than one layer for encoding and decoding (Stacked Autoencoder or Deep Autoencoder) can have advantages. It facilitates the progressive dimensionality reduction in the case of encoding, and progressive reconstruction in the case of decoding. Stacked Autoencoders achieve better representations of the input data in the latent vector.

In Undercomplete Autoencoders, the dimension of the latent vector is smaller than the size of the input. One of the most common applications of this type of Autoencoder is the reduction of the input dimensionality. Furthermore, this means that only those features that differentiate one input from the rest are maintained in the latent vector. The task being performed is an automatic feature extraction.

Denoising Autoencoders are used to remove noise from input data. One of the side effects of removing this noise is a better representation of the data in the latent vector (i.e. better feature extraction) [8]. The objective of this type of network is usually to obtain a good representation of the input data.

The following are some combinations and an application that are used in the project.

- **Stacked Denoising Autoencoder (SDAE)**: This type of network combines the Stacked Autoencoder and the Denoising Autoencoder, achieving an Autoencoder that takes advantage of both models.
- **Convolutional Autoencoder (CAE)**: This type of network uses convolutional layers. When images are used as network input, it makes sense to use this type of layer.

- **Autoencoders applied to image segmentation**: Autoencoders can also be used to perform segmentation tasks. In this type of network, an image is introduced at the input and the segmented image is introduced at the expected output. By performing the training, it is possible to obtain a network that learns how to perform the segmentation [9].

  A similar task is the **extraction of an area of interest** from the input image. By inserting the input image in the expected output but only keeping the elements of interest, it is possible to obtain a network that learns to isolate those elements of interest in an image.

### E.2.3. Image processing

OpenCV has been used as a library for Computer Vision. Mainly color masks have been applied to correctly identify an area of interest in an image based on color similarity. To perform this task, it has been necessary to select an appropriate color model.

The HSV [10] color model is the most used to make color selections in an image since it is based on intuitive concepts. The parameters of this model are hue (H), saturation (S) and value (V). The hue or tint represents the color spectrum. Saturation is the parameter that determines the purity of the color. Value represents the illumination of the color. The great advantage of this model is the possibility of comparing colors using only the hue value (H).

The Chroma Key [11] technique will also be used, which consists of substituting a color in an image or video and replacing the area occupied by that color with another image or video.

### E.2.4. Related works

#### *E.2.4.1. End to End Learning for Self Driving Cars*

Bojarski et al. [12] proposed a method of learning vehicle control based only on the input image using a Convolutional Neural Network. In this work the images are obtained from a front camera placed in a vehicle.

The neural network performs a mapping of the pixels of the driving images directly to steering commands. This method has the advantage that the driving system is simplified since it is the convolutional network that automatically learns to understand the environment and planning.

Without the need for prior processing, the network was able to learn the necessary characteristics of a road, with or without lines, just by providing the label of the steering control given by a human driver.

With this work they managed to demonstrate empirically that a CNN is able to learn the complete task of following a road or a path without applying techniques to understand the environment in an explicit way.

#### *E.2.4.2. Deep Learning Technique-Based Steering of Autonomous Car*

Y. Yang et al. [13] proposed a Stacked Autoencoder whose objective is to learn the necessary characteristics of driving in order to finally achieve autonomous vehicle steering.

The training is done in two phases. In the first phase, a Stacked Autoencoder is pre-trained using the same image both at the input and output. In this training the network learns to extract the features of the driving circuit.

In the second phase an output layer with three neurons is added to the Stacked Autoencoder. Training in this phase is done with three turn labels: left, straight or right.

Their results show that the vehicle is able to drive around the circuit even when making changes in the environment, which demonstrates that the network is able to generalize and to focus on features of the circuit itself and not the environment itself.

In relation to the work done in this article, our own work provides certain contributions. In particular, a chroma-based artificial stage will be used to facilitate the separation of the lines of the road from the rest of the elements, so that only the features corresponding to those lines are be extracted.

In addition, this stage will allow the generation of augmented images, which allows a virtual training of the network in realistic scenes.

Another major contribution is the use of a more elaborate Autoencoder, namely a Denoising Stacked Convolutional Autoencoder, which by adding noise to the input data (Denoising) has made it possible to improve the robustness of the trained model.

## E.3. ANALYSIS AND DESIGN

### E.3.1. Problem description

The main objective is to develop a small-scale vehicle that learns to follow a route delimited by two road lines. This problem is divided into two tasks, the extraction of features from the road lines and the driving learning.

The input of the autonomous driving system will be an image from a front-facing camera placed in the vehicle. The system must extract features from the road lines present in the image and use them for driving. Vehicle control learning will be based on the use of artificial neural networks through supervised learning. The output of the driving system must correspond to a value related to the direction of the vehicle (steering value). A diagram of this system can be seen in Figure E.2.

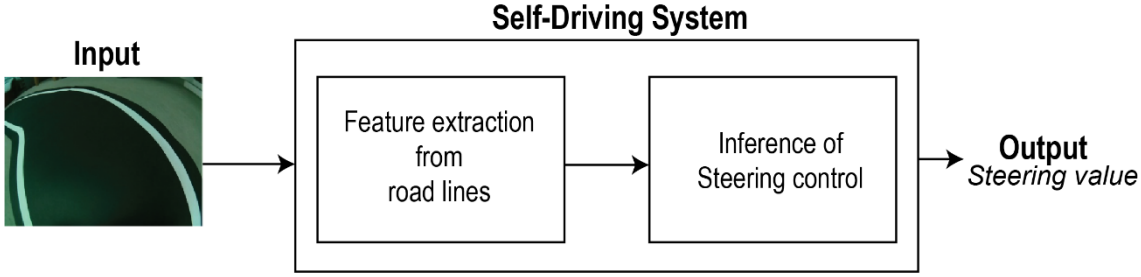



Figure E.2: Self-Driving system schema.

Dealing with the task of extracting the lines from the road in a real scene would be very complicated since it is necessary to differentiate these elements from the rest in the scene. To make this task possible, we propose to train an Autoencoder in an artificial stage that allows to differentiate the elements. In this way it will be possible to generate augmented training images from real scenes for the input, and data with only the road lines for the expected output of the Autoencoder. After the training, the Autoencoder can be used in the real stage to isolate the lines.

A small-scale vehicle will be developed to deal with the tasks of the project. The necessary processing will be done on an SBC (Single Board Computer), i.e. a complete computer in a single circuit. Except for the training of the neural networks, all processing tasks will be performed in the vehicle.

Since a realistic driving style is wanted, the steering will be controlled by the front wheels via a steering axle. The thrust of the vehicle will be developed by two electric motors on the rear wheels.

To solve these tasks, it is necessary to develop a control and training data collection system. This system must offer supervised control of the vehicle remotely (through a control device with real-time video). This system will also be responsible for building training data sets composed of the driving image and a value indicating the state of the vehicle's steering.

### E.3.2. Initial design phases

The development of the project has been divided into five phases. The four initial phases focus on the learning and the incremental understanding of the necessary knowledge to reach the objectives established in the project.

The final phase deals with the development of the final vehicle and the experimentation to verify its correct operation.

Here we summarize the tasks of the first four phases.

- **Phase I - Object tracking**: In this phase, Computer Vision techniques are used to track an object with a camera using a Raspberry Pi. In addition, a reduced scale vehicle is used by making the direction of the vehicle turn towards the position of the detected object.
- **Phase II - Experimentation with Deep Learning**: In this phase, experimentation in different domains are performed to become familiar with the necessary libraries and tools. The domains of Dogs vs Cats [14], Sign Language MNIST [15] and MNIST [16] through Autoencoders are addressed. An approach to the task of extracting features from road lines is also performed using a simulator. This approach has allowed to get in touch with the work to be developed.
- **Phase III - Control and training data collection system**: This phase deals with the technical aspects necessary to develop a control and training data collection system for the scaled vehicle. This system is described in the final design section.
- **Phase IV: Extraction of features from the road lines (First environment)**: In this phase, a simplified approach to a real environment is made. The working environment of this phase is composed of an artificial stage that allows the application of the Chroma Key technique and a road composed of two lines. This road is duplicated, so that training can be done in the artificial stage and then tested in the target stage. This task (on the final environment) can be seen in more detail in the experimentation section.

### E.3.3. Final design

The final design of the elements needed to deal with the project tasks is described below.

#### E.3.3.1. Control and training data collection system

The developed system performs several tasks:

- Create a wireless connection between a computer and the vehicle
- Show a video on the computer with images captured in real time by the vehicle.

- Control the vehicle using a steering wheel connected to the computer. The Logitech G29 Driver's Kit is used.
- Save the images of the real-time video and the steering labels that will allow to make training data sets.

A diagram of this system can be seen in Figure E.3. In **Video 3.5** this system can be seen in operation.

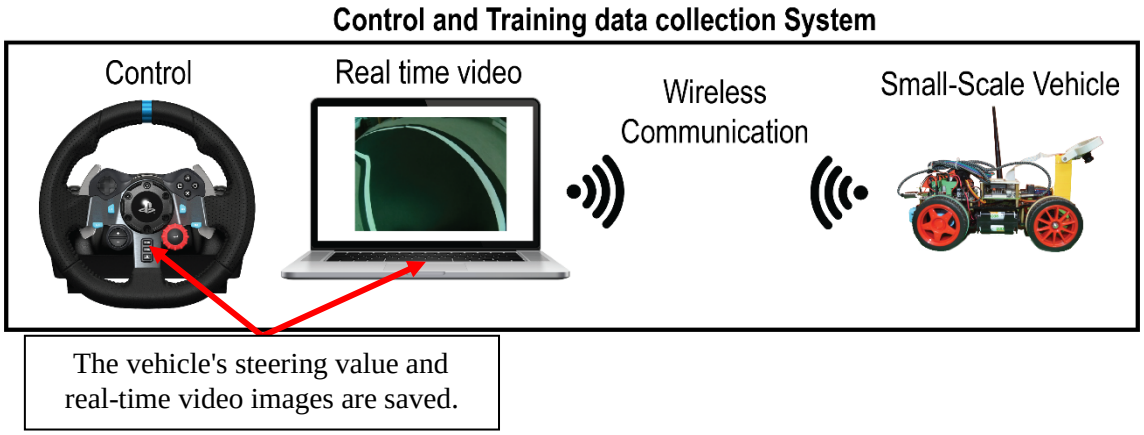


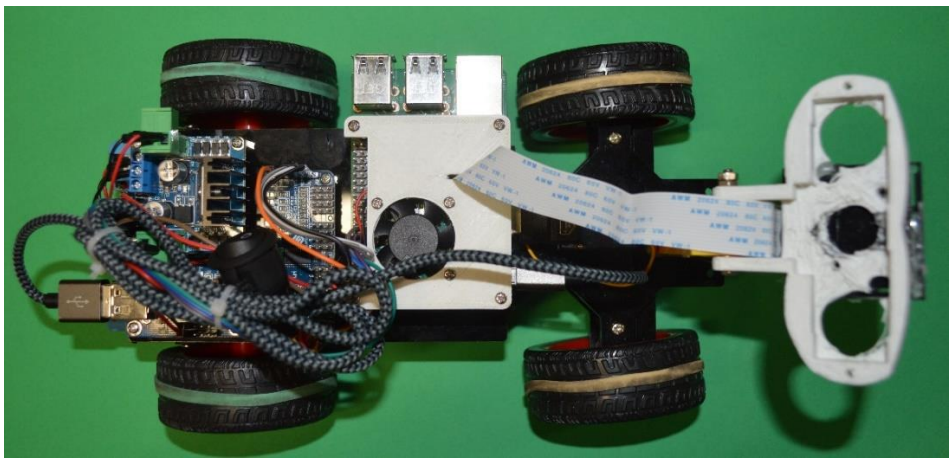

Figure E.3: Control and training data collection system scheme.

### E.3.3.2. Final vehicle

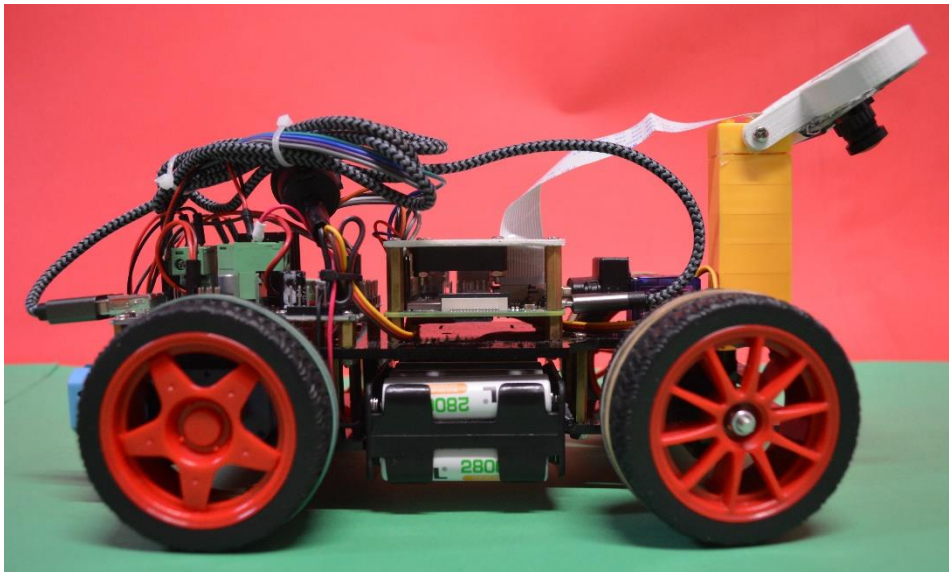
The final vehicle prototype consists on the following components:

- Processing is done in the SBC Raspberry Pi 3 B+.
- A USB wifi antenna is used for a more reliable communication.
- The Pi Camera rev 2.2 with a 160° viewing angle is used as the front camera.
- Eight AA Nickel - Metal Hydride batteries and a XL1509 based voltage regulator with a constant output of 5 V and 2 A are used to power the system.
- The L298N module is used to drive the motors.
- Two 1:90 TT Motor are used as the vehicle's traction motors.
- The servo motor that controls the steering is the SG92R.
- The PCA9685 module is used to control the PWM signals needed to regulate the speed and steering of the vehicle.

The final vehicle can be seen in Figure E.4.



(a) Top view



(b) Side view

Figure E.4: Views of the final vehicle prototype.

The vehicle's connection diagram can be seen in Figure E.5.

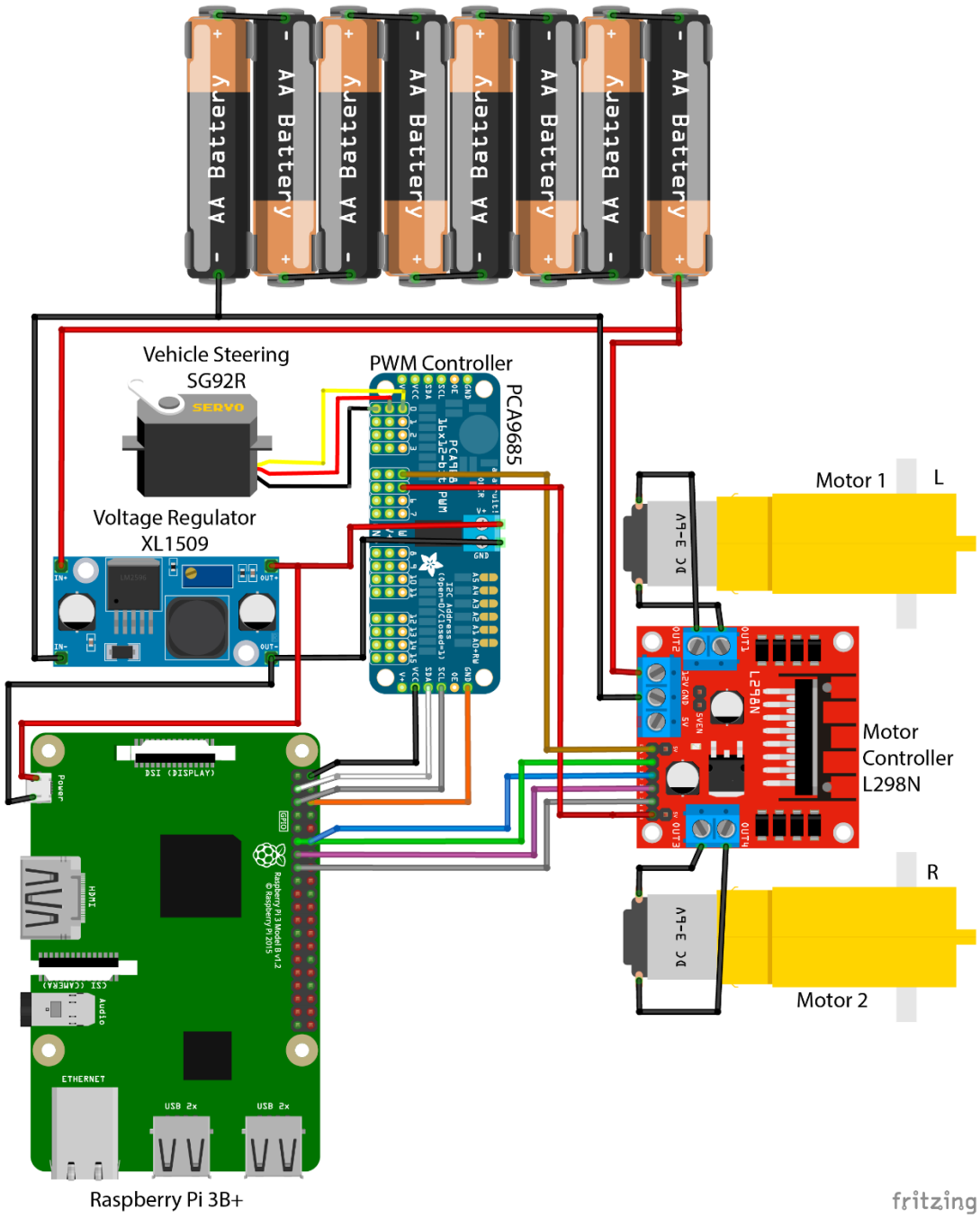



Figure E.5: Vehicle connection schema.

### E.3.3.3. Working environment

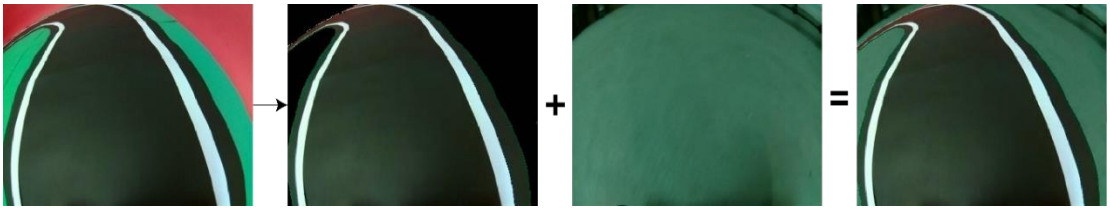
To carry out the task of extracting features from the road lines, an artificial scenario based on chromas is proposed. The materials have been appropriately selected to make it possible to apply the Chroma Key technique. Thanks to this technique it will be possible to extract the lines from the road correctly. Contrary to what would happen in a real environment, due to the difficulty of differentiating the lines from the rest of the elements.

The working environment is composed of the following elements:

- **Driving circuit**: The developed circuit contains the lines of the road on black cardboard that imitates asphalt. The track is totally closed to facilitate the training

task since driving is done without the need to reposition the vehicle, that is, in a cyclical manner.

- **Target Stage**: This is the final stage (the "real" world stage) in which the task of learning should be performed. In the case of this project, this stage is called *sótano*.
- **Chroma Stage**: Allows the extraction of the road lines, and the subsequent generation of augmented images. This artificial stage is composed of easily separable colors. Placing the driving circuit inside this stage, it is possible to capture images and process them to eliminate everything except the road or the road lines. Once the road is isolated, it can be superimposed on the images of the target stage and used for training. Figure E.6 shows an example of this process.



(a) Image captured on chroma stage | (b) Image processed. Isolated road. | (c) Image captured on target stage | (d) Resulting augmented image

Figure E.6: Augmented data generation process flow.

Figure E.7 shows the elements and the different stages.

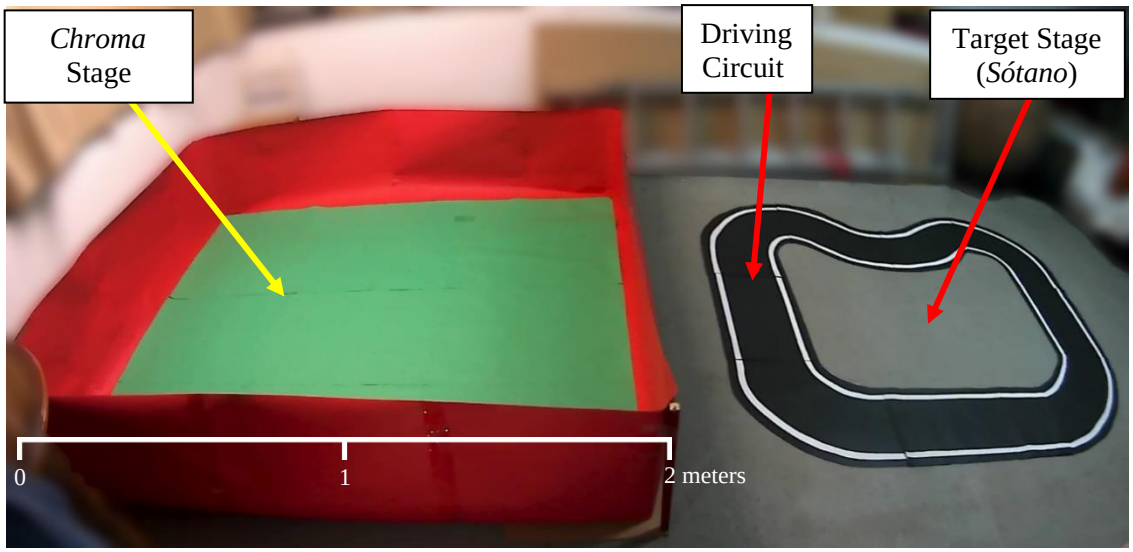



Figure E.7: Arrangement of the elements in the final working environment.

This image shows the final environment in which the training has been done. The driving circuit can be placed on the chroma stage (left) or on the target stage (*sótano*) depending on what is needed. A scale line is included to know the approximate size of the elements.

## E.4. EXPERIMENTATION

This section will deal with the two main tasks of the project.

At first, the experiments to isolate the road lines and to obtain the corresponding feature vector will be described.

Then the experimentation to deal with the task of learning to drive autonomously will be described.

The steps that are needed for both tasks are shown in **Video 5.1**.

### E.4.1. Extraction of features from road lines

This task consists in training an Autoencoder that can ignore the accessory elements of an image, that is, extracting the lines of the road. The latent vector of this network will contain the values necessary to represent these lines. The following setup will be used to train the network:

- The car is driven on the circuit inside the chroma stage while its camera captures images.
- **Input**: Image captured in the chroma stage, removing everything except the road lines and asphalt. This task is simplified and made possible using the Chroma Key technique. The result is superimposed on images of the target stage (*sótano*).
- **Expected output**: Image captured in the chroma stage, removing everything except the lines.

In this way, the Autoencoder is expected to learn to extract only the road lines in images that contain multiple elements.

Various approaches have been used for this task during the project. Phase IV uses a slightly more simplified environment than the final one. Poor reconstructions were initially achieved. However, by applying the noise input addition technique to the training data, it has been possible to obtain good results. The improvement can be seen in the **Video 4.5**.

The structure and details of the Autoencoder finally used are as follows:

- Latent vector of 32 values.
- Four convolutional layers, each one followed by a MaxPooling layer that reduces the size of the data. The successive convolutional layers use 32, 64, 128 and 256 filters, respectively.
- Noise addition in the input data using a Gaussian distribution with a standard deviation of 0.4.
- Training takes 40 epochs. The Batch Size used for training is 32.

The results of this task have been satisfactory, allowing a good reconstruction of the circuit in the target stage. A diagram of the network used and an example of reconstruction can be seen in Figure E.8. The reconstruction of a test sequence in the target stage can be viewed in **Video 5.3**.

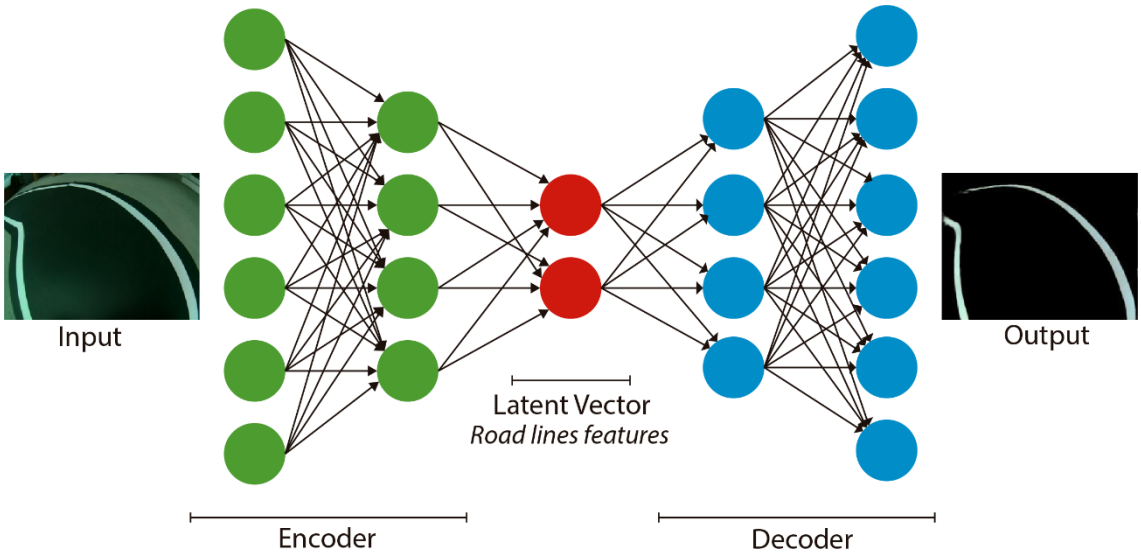



Figure E.8: Graphical representation of the line extraction network (*Autoencoder*)

### E.4.2. Learning to Drive

This task addresses the learning of autonomous vehicle steering control. The neural network responsible of this part rather than using an image as input, relies on the features extracted by the Autoencoder. The steering training is done once a reliable Autoencoder is available, so it is done on the driving circuit placed in the target stage (*sótano*).

To perform this task an MLP is used, the structure of the network finally used will be described at the end of this section. This network is trained using driving sequences in which the captured image and the value of the steering axis are stored. The training will be done with the features of the road lines as input and the direction value as expected output.

The results obtained in the first experiment are not very satisfactory. The causes of this problem are attributed to the inference time and the speed of the vehicle. The SBC manages to infer about 2.5 images per second, which is a very small amount. The speed of the vehicle is the minimum allowed by the motors; it would be interesting to use other motors with a more adequate speed range.

An attempt is made to reduce the inference time. This is done by reducing the size of the input image from 320x240 to 160x120 and retraining the Autoencoder. This provides an agent capable of updating the direction of the vehicle about 5 times per second.

Although the autonomous driving improves, the vehicle was still losing track at some points of the circuit. In the supervised driving phase, the driving style had not been adequate enough. To solve this, the driving sequences are repeated again, using this time a much more careful and smoother driving style, especially when taking the bends.

Another improvement was the installation of a pair of traction motors that offer a more adequate speed range. This detail had to be taken into consideration at the final phase once the inference time of the final driving agent was known.

Finally, a satisfactory agent is obtained. This one can complete the circuit following the route delimited by two road lines without going out. In the **Video 5.4** it is possible to visualize the vehicle driving around the circuit in an autonomous way in both directions.

The densely connected network that has been used has the following characteristics:

- 5 densely connected layers that alternate with dropout layers. The densely connected layers have 512, 256, 128, 64 and 1 nodes respectively.
- Each dropout layer has a probability value of 0.2.
- All layers use the ReLU activation function except the last one, which corresponds to the output, so it has a linear activation function.
- The training lasts for 20 epochs.

A diagram of the complete neural network can be seen in Figure E.9.

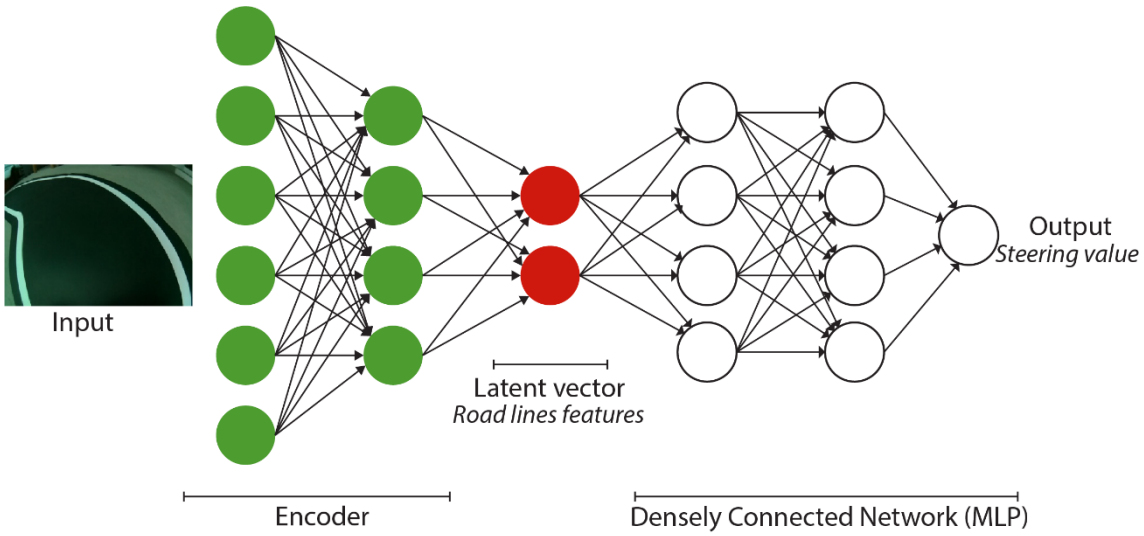



Figure E.9: Graphical representation of the complete conduction network.

## E.5. VALIDATION

Many tests have been performed to confirm that the requirements defined in the project have been satisfied. In addition, other tests have been performed to determine the reliability of the developed autonomous agent.

These last tests have allowed to know the level of robustness and generalization reached, the most remarkable ones are presented below. A section called **Validation** has been included in the project repository (E.8. REPOSITORY OF THE PROJECT) with pictures and videos of the result of some of the tests carried out in this chapter.

- The agent has shown great robustness to lighting changes in the stage, as well as to shadows or slight variations of the light source.
- The agent has shown great robustness to changes in the stage. The appearance of obstacles on the road that partially cover the road lines has not been a problem. With the occasional occlusion of one of the lines, the agent has been able to drive guided by the other line.
- The agent successfully completed a circuit with a different route for which he had been trained initially.

In conclusion, the developed autonomous agent has shown a good level of robustness and generalization.

## E.6. CONCLUSIONS AND FUTURE WORK

In short, and as a general conclusion, the approach discussed in this project is a functional possibility for autonomous steering control of a small-scale vehicle.

The approach proposed in this project of using an Autoencoder has been successful. More techniques or variants of Autoencoders have been used, until obtaining an architecture equivalent to a Denoising Stacked Convolutional Autoencoder. It has been possible to verify how the addition of noise in the input data (Denoising) has been fundamental to train the network with augmented images and to make it work properly in the target scenario.

In relation to the above, using the chroma or artificial stage is what made possible to differentiate the lines of the road from the rest of the elements of the scene. In this way it has been possible to obtain a vector of features related only to these lines. In addition, this scenario has allowed the generation of augmented images useful to train the agent and use it later in the driving stage (target stage, *sótano*).

The driving learning from the extracted features has been successful. It has been possible to observe how the developed agent imitates the behaviour of the captured driving data. An important conclusion that can be drawn is that the driving style of the training is quite important. With careful driving by the user, the resulting agent has been able to complete the circuit successfully.

Since the learning phase has been divided into two tasks, feature extraction and driving learning, the development of the system has been more flexible, making it possible to approach the work incrementally and reducing the required training time. Once the Autoencoder has been trained, training the steering network is a faster process. In this way, it is possible to train different driving styles more quickly. An approach based on convolutional neural networks would have involved retraining the entire network.

As for the vehicle control system, control has been achieved with some realistic aspects. Both the steering control carried out with a steering wheel, and the visualization of the road, carried out from a front view in real time, quite reflect reality.

For future work, it is proposed to deal with the task of extracting features from road lines in the real world, making them functional on realistic roads. It would be necessary to generate real data sets to isolate the lines. Since isolating the lines manually or using Machine Vision techniques can be tedious or complex, it is proposed to improve the developed Chroma scenario so that it is possible to create more realistic augmented images.

The quality of the hardware could also be improved. One possibility would be to use a Nvidia Jetson Nano as the onboard computer, improving the processing power.

Adjusting the camera's angle of elevation allows the agent to visualize parts of the route earlier or later, and therefore make decisions with more or less anticipation. This aspect allows that an agent trained at a certain speed can be used at different speeds by modifying that angle. This could be another line of future work.

As the last possible future work, it is proposed to improve the realism of the vehicle control system. A 360º camera could be installed in the vehicle and a virtual reality helmet could be used so that the user has a realistic perspective of driving.

This project was developed under a grant from the Ministerio de Educación y Formación Profesional of Spain, as part of its 2019/2020 scholarship program (Beca de Colaboración).

## E.7. BIBLIOGRAPHY

[1] D. A. Pomerleau, 'ALVINN: An Autonomous Land Vehicle in a Neural Network', in *Advances in Neural Information Processing Systems 1*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1989, pp. 305–313.

[2] M. Williams, 'PROMETHEUS-The European research programme for optimising the road transport system in Europe', in *IEE Colloquium on Driver Information*, 1988, p. 1/1-1/9.

[3] 'Technology', *Waymo*. https://waymo.com/tech/ (accessed Jun. 25, 2020).

[4] 'Putting Self-Driving Cars On The Roads | General Motors'. https://www.gm.com/our-stories/self-driving-cars.html (accessed Jun. 25, 2020).

[5] 'Autopilot'. https://www.tesla.com/autopilot (accessed Jun. 25, 2020).

[6] 'Waymo expands driverless car testing to Florida', *VentureBeat*, Aug. 20, 2019. https://venturebeat.com/2019/08/20/waymo-expands-driverless-car-testing-to-florida/ (accessed Jul. 02, 2020).

[7] J. Sietsma and R. J. F. Dow, 'Creating artificial neural networks that generalize', *Neural Netw.*, vol. 4, no. 1, pp. 67–79, Jan. 1991, doi: 10.1016/0893-6080(91)90033-2.

[8] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, 'Extracting and composing robust features with denoising autoencoders', in *Proceedings of the 25th international conference on Machine learning*, Helsinki, Finland, Jul. 2008, pp. 1096–1103, doi: 10.1145/1390156.1390294.

[9] V. Badrinarayanan, A. Kendall, and R. Cipolla, 'SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation', *ArXiv151100561 Cs*, Oct. 2016, Accessed: Jun. 03, 2020. [Online]. Available: http://arxiv.org/abs/1511.00561.

[10] D.D. Hearn and M.P Baker. Computer graphics with OpenGL. 3rd Ed. Pearson Prentice Hall, 2004.

[11] '¿Qué es un Chroma Key?', *Monsuton*, Aug. 29, 2019. https://www.monsuton.com/croma-key/ (accessed Jul. 02, 2020).

[12] M. Bojarski *et al.*, 'End to End Learning for Self-Driving Cars', *ArXiv160407316 Cs*, Apr. 2016, Accessed: Apr. 24, 2020. [Online]. Available: http://arxiv.org/abs/1604.07316.

[13] Y. Yang, Z. Wu, Q. Xu, and F. Yan, 'Deep Learning Technique-Based Steering of Autonomous Car', *Int. J. Comput. Intell. Appl.*, vol. 17, no. 02, p. 1850006, May 2018, doi: 10.1142/S1469026818500062.

[14] 'Dogs vs. Cats'. https://kaggle.com/c/dogs-vs-cats (accessed Apr. 29, 2020).

[15] 'Sign Language MNIST'. https://kaggle.com/datamunge/sign-language-mnist (accessed Apr. 29, 2020).

[16] Y. LeCun and C. Cortes, 'MNIST handwritten digit database', 2010, Accessed: Apr. 14, 2020. [Online]. Available: http://yann.lecun.com/exdb/mnist/.

## E.8. REPOSITORY OF THE PROJECT

References have been made to resources in a repository dedicated to this project. This repository is called TFGAutonomousCar and is hosted by GitHub, at **https://github.com/javiercorrochano/TFGAutonomousCar**.

The following nomenclature has been used to refer to videos: Video X.Y, where X is the number of the phase to which it belongs, and Y is the video number within that phase.